AN EFFICIENT MODEL OF DYNAMIC TASK SCHEDULING FOR DISTRIBUTED SYSTEMS

Arif Ghafoor
Department of Electrical and Computer Engineering,
Syracuse University, Syracuse, NY 13244.

Ishfaq Ahmad
School of Computer and Information Science,
Syracuse University, Syracuse, NY 13244.

ABSTRACT: In decentralized task scheduling models, the decision to transfer load from the most heavily loaded nodes to the most lightly loaded nodes does not always prove beneficial. Furthermore, the overhead due to information exchange and load transfer needs to be reduced and the system topology should not impose any constraints on the performance. We propose a distributed task scheduling model by using a simple processor architecture and a heuristic scheduling algorithm based on small message exchanges between nearest neighbors. Through an extensive simulation study, we analyze the proposed model by taking into account a wide range of practical issues. Comparison with other schemes reported in the literature reveals the superiority of our model in terms of various performance measures. The model incurs a reduced overhead due to the information exchange and exhibits an adaptive nature to network topology.

1 INTRODUCTION

In order to obtain a high performance from a distributed system, the operating system must be equipped with an efficient strategy for scheduling workload. The workload for a distributed system is characterized by tasks [1] which can be independent application modules or inter–related sub modules of an application. If an application comprises a number of communicating tasks, then the communication overhead needs to be reduced and the precedence constraints have to observed for a faster response time. Scheduling strategies that assume a priori knowledge of the task characteristics are termed as static scheduling strategies [3]. Static [9] scheduling schemes prove effective in initial application to system mapping but are less beneficial in a time dependent environment where load on individual processors can fluctuate.

Dynamic scheduling strategies, which do not assume a priori knowledge of the work load, take into account the unpredictable fluctuations in load patterns across time and space [12]. An efficient dynamic scheduling strategy balances the workload by transferring some load from the heavily loaded processors to lightly loaded processors. A large number of such strategies with various decentralized algorithms [5][8][15] have been suggested in the literature. Most of these strategies employ heuristic approaches [6], and probabilistic [15] and queuing models [11]. Demand driven models are discussed in [10] where lightly loaded processors initiate requests for load. Load balance in distributed database systems has been explored in [16]. A number of algorithms are compared in [17] using a trace driven simulation model. Although decentralized models have the potential advantages over the centralized models[17], they can incur large overhead due to information exchange and task migration. The optimal scheduling decisions, in the decentralized algorithms depend on the accuracy and amount of state information and are hard to obtain.

In this paper, we present a model of dynamic task scheduling and load balancing for distributed systems. The proposed model is intended for any interconnection structure with the size ranging from a few nodes to hundreds of nodes. Contrary to the intuitive notion that the load should be migrated from the most heavily loaded processor to the most lightly processor, we show that if every processor *equalizes* its load within its *neighborhood*, the performance can be significantly increased. The first objective of our study which distinguishes it from others is to overcome instability inherent in scheduling decisions. Instability of scheduling decisions [1][5][6] is one of major problems in a decentralized environment where the system state changes rapidly. The second objective is to consider the physical locations of the system nodes for collecting state information without increasing the complexity of message exchanges. Confining the message communication between nearest neighbors results in a symmetric scheduling algorithm which can be applied to any system topology. Thirdly, we show that a subtle variation in various parameters can have dramatic impacts on the performance. In this paper, we identify those parameters, consider their effects on various performance measures and prove the efficiency of the model under varying circumstances. That makes our model general enough to be applicable to more practical environment. Variations are introduced in the main scheduling algorithm to allow investigations into the effects of information collection disciplines and the choice of node for load transfer. The proposed algorithms were analyzed through an extensive simulation study and compared with a no load balancing environment as well as load balancing schemes suggested in the literature [5][17]. The proposed algorithms are shown to produce much better performance in terms of response time, communication overhead, overhead incurred due to information exchange and processor utilization even under heavy loading conditions. They are also shown to be insensitive to the underlying network structure.

In the following section, we present a system architecture model. We discuss the network and a node model with a brief explanation about its functional units. Section 3 gives a description of the algorithm which employs simple heuristic techniques combined with a queuing model. Next, we discuss the four variations in the algorithm. Section 4 discusses the simulation model and results are given in section 5 and 6.

2 DISTRIBUTED SYSTEM MODEL

A distributed system can be modeled as a collection of processing nodes connected by a communication network. The nodes contain general purpose resources such as memories, processors, databases etc. and provide an execution environment for the tasks entering into the network. The network is assumed to be homogeneous where all nodes are identical. The model is described further in the following sections in terms of components which are used by the task scheduling algorithms.

2.1 Network Model

The network of a distributed system provides high speed communication channels for direct communication between any pair of nodes. A node needs to communicate only with its immediate neighbors and take decisions accordingly. The network topologies, considered for simulation study are described in sections 4 and 6.4.

2.2 Node Model

A node in our model is assumed to be an independent processing element with its own local memory and operating system. Each node is connected to other nodes through bidirectional high speed communication channels. For the scheduling algorithm a node needs a set of functional modules which are Information Collector/Dispatcher, the Task Scheduler, a set of queues for holding tasks and a set of Mailboxes for receiving control messages and tasks. Figure 1 shows the overall functional model of the node in terms of their interactions. We now briefly describe the function of each module.

2.3 Information Collector/Dispatcher

The ICD (Information Collector/Dispatcher) unit obtains information about the local load and that of neighboring nodes and keeps this information in a table which resides in a fast memory named as Link Load Table. In the proposed algorithms, such information represents the load status of a node and it provides an estimate of the total service time of all the tasks which have been assigned to that node for the final execution. The estimated service time of a task, which is assumed to be known, is a rough prediction of the service demand of a task including CPU cycles, memory demand and I/O requirements. The ICD dispatches the information about the local load, which is the sum of the estimated service times of tasks in the local execution queue, to the neighboring nodes for their "view" of this particular node's load. These messages about the load status are called Link Load Views. The exchange of Link Load Views can be done by message passing in two ways. When the Task Scheduler needs to schedule a task, the ICD collects its Link Load Views from each ICD of the neighbors. Alternatively, the ICD at each nodes in the system can dispatch its local load status to all neighbors periodically.

2.4 Task Scheduler

The TS (Task Scheduler) is responsible for deciding whether an incoming task is to be serviced locally or be transferred to a neighbor. For scheduling, the TS executes the scheduling algorithms (described in section 3.1) by consulting

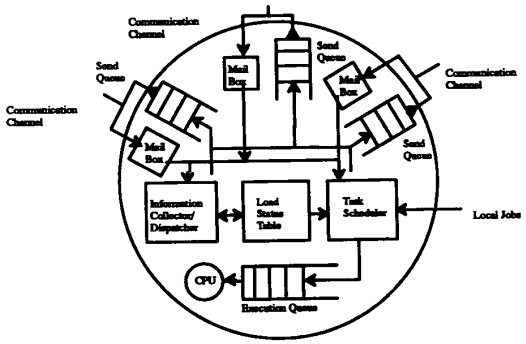

Figure 1. Node Model

the Link Load Table. If the TS decides that a task is to be executed locally, it schedules that task at the end of the execution. Otherwise the task is put in the send queue to a neighbor which is considered a suitable choice.

2.5 Send Queue

For each communication link, there is a send queue that maintains all outgoing traffic. The reason to maintain a queue for each link is because tasks are of different sizes and consequently need variable communication times. Moreover, the contention at the network channel is another factor that affects the routing. Send queues are served on the FCFS basis.

2.6 Mail Boxes

Each communication link maintains a dedicated mailbox for receiving incoming packets. Tasks arriving from outside world and from mailboxes are handled with equal priority.

2.7 Execution Queues

All tasks which are to be executed locally are kept in the execution queue. The CPU, serving on the FCFS principle, keeps itself continuously busy as long as the queue remains non empty. We assume that both CPU and TS operate concurrently.

3 THE TASK SCHEDULING ALGORITHMS

In this section we first present the basic algorithm and then discuss the four variations. Whenever the Task Scheduler of a node receives a task (locally generated or migrated from another node), it executes the scheduling algorithm and decides whether the task should be executed locally or it is to be routed to a neighboring node. The notations that we use are described along with their meanings in Table 1.

The system topology is represented by an undirected graph, G = (V, E) where V = {1,2,3,....N} represents the set of nodes and E = {1,2,3,....L} is the set of links between the nodes. The neighbor of a node is defined as a node at one hop distance i.e. directly connected. The degree of i–th node is d_i which is also the number of neighbors of that node. Note that d_i is a constant for all the nodes in a symmetrical (regular) network topology. Let T_j^k be the estimated service time of task k at node j. If j is the neighbor of node i and node j has Q_j tasks scheduled to run in its local queue, then the quantity $\sum_{k=1}^{Q_j} T_j^k$ is taken as "Link Load View" of neighbor j seen by node i and is denoted as V_i^j. As mentioned earlier this value is obtained by the Information Collector/Dispatcher for each link. Let T_i^0 be the expected service time of the newly arrived task at the i–th node.

3.1 The Main Algorithm

For the algorithm, the i–th node takes the following steps upon arrival of a task.

Step 1. Calculate the Local Load View as

$$V_i^0 = \sum_{k=1}^{Q_i} T_i^k$$

If it is zero i.e. local queue is empty, put the task in local queue.
Otherwise continue.

Step 2. Check the transfer tag of the arrived task.
If its value has reached R_t, put the task in the local queue.
Otherwise continue.

Step 3. Obtain Link Load Status information
for each neighbor j of i, calculate Link Load View for link e_i^j as

Table 1: Notations and their meanings

N	Number of nodes in the network.
L	Number of edges in the network.
e_i^j	link for node i to neighbor j.
d_i	Connectivity of node i.
Q_j	Number of tasks in the Execution Queue of node j.
T_j^k	Estimated Service time of task k at node j.
V_i^j	Load View of node j seen by node i through link e_i^j.
V_i^0	Local Load View of node i.
R_t	Transfer Limit of a task
TH_i	Dynamic Load Threshold of node i.
T_u	Time period for Periodic Link Load update.
λ	Arrival Rate per node.
μ_s	Execution Service Rate.
μ_c	Communication Service Rate of the network channels.

$$V_i^j = \sum_{k=1}^{Q_j} T_j^k$$

Step 4. Calculate the Dynamic Load Threshold

$$TH_i = \frac{\left[\sum_{j=1}^{d_i} V_i^j + V_i^0\right]}{d_i + 1}$$

Step 5. If $V_i^0 + T_i^0$ is less than TH_i,
put the task in the local Execution Queue. Otherwise continue.

Step 6. Select link e_i^j such that its V_i^j is the minimum and less than TH_i
Increment the transfer tag of the task by 1.
Put the task in j–th Send Queue.

3.2 Design Objectives for the Proposed Algorithms

The proposed algorithm is fully distributed and simple in nature. The algorithm and the associated node model is applicable to any network topology. Also, the proposed approach has a number of characteristics which meet the following important objectives.

(1)Transfer policy: The decision to execute a task locally or remotely is called Transfer Policy [5]. Approaches using a threshold policy have been suggested for this purpose [5][17] where a node's load is transferred to another node only when it exceeds a predefined threshold. In a dynamic environment, where system states are unpredictable, predefined threshold limits may not be suitable for all loading conditions. In the proposed algorithm, threshold limits are dynamically determined by calculating a node's load relative to its neighbors.

(2) Location Policy : Decisions about choosing a node for load transfer are determined by the location policy. Selection of a target node requires determination of a load index defining the status of that node. As opposed to the common practice of considering queue lengths as the estimation of load status [17], our location policy takes into account the notion of *queue weight* which represents the sum of the estimated service times of all tasks in the ready execution queue. In a realistic environment, tasks are of considerably different sizes in terms of their service requirements. The accumulative service time of a queue of longer length filled with small tasks may be less than a shorter queue with larger tasks. Therefore, *queue weight* as a load index for the location policy is a more precise measurement than merely considering the number of tasks in the queue.

(3) Amount of Information Exchange : The advantage of reducing the amount of system state information was stressed earlier. Many complex algorithms have been suggested for maintaining system state information [1][4][5]. These schemes are attractive for systems comprised of small numbers of nodes, but as the system size increases, the collection of information at a global level in a rapidly changing environment poses serious constraints on scheduling decisions.

(4) Prevention of Task Thrashing : The tasks are allowed to migrate in the network before finding a suitable node. Initially a node may transfer a task to its neighbor considering it a suitable choice. However, the neighbor may transfer the same task to another node finding. As a result, the tasks can go into *thrashing state* where they may make a large number of migrations or even traverse cycles in the network topology without settling down at one node. To avoid *thrashing*, a transfer limit, R_t, for every task is required. This can be achieved by attaching a tag to each task. The value of this tag is increment by 1 every time the task is migrated. When this value reaches R_t, the task is executed at that node unconditionally.

(5) *Topological Independence:* As mentioned earlier, a distributed application should be independent of network topology [1][13]. This feature is present in our scheme owing to the nearest neighbor communication requirements for information exchange and load transfer. Therefore, the algorithm can be implemented on any network topology. The transfer and location policies in the algorithm are only dependent upon the number of links (degree) incident to each node.

3.3 Variations in Information Collection and Node Selection

In order to analyze the behavior of a distributed system in view of the above mentioned objectives, we consider four variations in the scheduling algorithm; the variations are related to information collection and node selection procedures. The resulting schemes use the same transfer policy but differ in location policy and information exchange. The original algorithm will be termed as *Fresh Update Best Selection (FBS)*. The rest of the algorithms are as follows.

FRS (Fresh update random selection): For this scheme, the Link Load Views are updated as in the case of *FBS* but the transfer policy in step–6 of the algorithm is implemented by selecting a link randomly among those links which have Link Load View less than TH (Dynamic Load Threshold). There are two advantages to this scheme. First the complexity for node selection is reduced. Second, it avoids overloading of very lightly loaded nodes which may appear with *FBS* scheme. The rest of the algorithm remains the same.

PBS (Periodic update best selection): In this scheme each node transmits the load status information periodically to all its neighbors after a fixed time interval T_u. The ICD of every node, therefore, periodically communicates with the ICD's of neighboring nodes and update their respective load status tables entries. The scheduling decision remains the same as described in the main algorithm.

PRS (Periodic update random selection): This scheme is essentially the same as *PBS* scheme except in step–6, the algorithm selects a link randomly among the links having Link Load Views less than the Dynamic Load Threshold TH_i.

4 SIMULATION MODEL AND RESULTS

The proposed algorithms were simulated to study their performance. The simulator was written in 'C' on an Encore Multimax containing 16 CPU's and 128 megabytes memory. Of prime importance was the accuracy of simulation. Each data point was obtained by taking the average of a large number of simulation runs with different random number streams. In each simulation run, 5000 tasks were generated. In addition, the simulator produces all results after reaching steady state. All results were obtained with 99% confidence interval with the size of the interval varying up to 5% of the mean value.

The main network considered for the experiment is a 35 node network with constant degree 4 and diameter 3. This network is shown in Figure 3 and it is called Odd graph. This topology is symmetric and possesses a rather high density in the sense that it is one of the known existing graphs with large number of nodes for the given degree and diameter [7]. Task generation was modeled as a Poisson process with average arrival rate λ tasks/unit–time, which was same for all the nodes. The execution and communication time of a task were exponentially distributed with a mean of $1/\mu_e$ time–units/task and $1/\mu_c$ time–units/task, respectively.

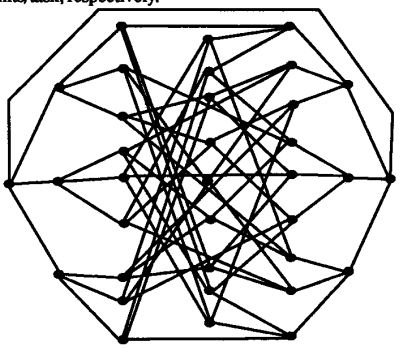

Figure 2. 35 Node Network with Degree = 4 and Diameter = 3.

5 Performance and Comparison of the Proposed Model

Comparisons of four proposed algorithms were carried out by changing various parameters such as system load, communication rate of the network channels, R_l (transfer limit), T_u (load status update period), network topology and the number of hosts. In order to compare the response times of the four algorithms on a unified basis, we chose parameters which generate similar environments. In order to analyze the relative improvements in the response time, the algorithms were compared with a no load balancing scheme implemented in the same environment. A comparison with another scheme reported in the literature as "Lowest" [17], "Shortest" [5] and more intuitively known as "best choice" scheme, is also included. We refer to this scheme as the *Least Selection* scheme since a task is *unconditionally* transferred to the most lightly loaded node among all the neighbors. If the local node is the least loaded among all neighbors, then the task is locally executed. Note that with the algorithms *PBS* (Periodic load update) and *FBS* (Fresh load update) a task is *conditionally* transferred to the lowest loaded node. In order to make a complete and fair comparison, we also implemented the "Least Selection" scheme with periodic and fresh load updates. From now onwards, we will refer to these algorithms as *PLS* (Periodic load update, Least selection) and *FLS* (Fresh load update, Least selection).

5.1 Response Time Comparison

Figure 3 shows the mean response times of all algorithms with varying the system load. All six algorithms yielded a substantial improvement in response time when compared to *NLB* (no load balancing) scheme. Curves for *PRS*, *PBS*, *FRS* and *FBS* run abreast in the lower load range. *FBS* yields better performance as the load increases. Essentially, the comparison is between three algorithms belonging to the fresh information update class and three algorithms belonging to the periodic information update class. The comparison scheme *FLS* performs better than the *PBS* and *PRS* algorithms at low loads but remains inferior to *FBS* because, under light loading conditions, the load status of neighboring nodes is less likely to change between the time the status is reported and the time a scheduling decision is made. *PLS* despite having same degree of accuracy in information gives higher response time when compared to the random scheme *PRS*. The performance of the proposed schemes remains substantially higher even at very high loads. For example, with utilization ratio exceeding 0.8, *FBS* yields 45% to 60% improvement over the no load balancing scheme and 25% to 28% improvement over *FLS*. *PRS* which produces the worst response time out of all proposed algorithms still gives an improvement of 2% to 25% over *FLS* and 40% to 30% improvement over *PLS*. All these comparisons reveal that the decision in transfer policy is more critical than the decision in location policy.

When comparing the proposed algorithms among themselves, the importance of the selection of a neighboring node becomes more apparent. *FBS* yields a better response time compared to *FRS* while *PBS* performs better than *PRS*. The effects of the accuracy of the load status information inherent in each algorithm are obvious from the curves shown in Figure 3 where both fresh update (*FBS* and *FRS*) algorithms outperform both periodic update (*PBS* and *PRS*) algorithms.

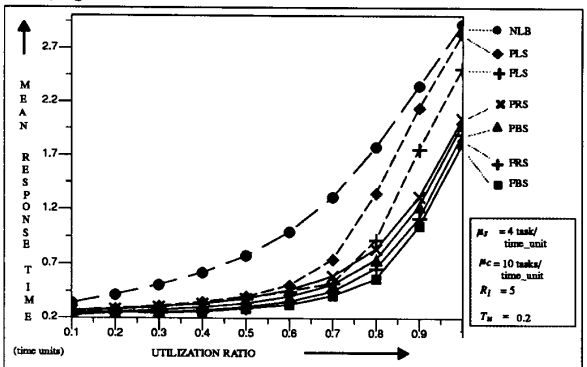

Fig. 3. Mean response time versus system load

5.2 Analysis of Load Distribution

The standard deviation of the total utilization of all the nodes is an estimate of "smoothness" of the load distribution over the whole network. In other words, it represents an variations in actual busy times of all the nodes relative to the global average. The standard deviations of node utilization for the six algorithms along with no load balancing scheme are shown in Figure 4.

The curve for no load balancing remains essentially constant for low arrival rates which indicates that there is a considerable number of nodes which remain idle while their neighbors may be heavily loaded and busy. This results in higher variations of load (and hence in busyness of nodes) across the whole network. This discrepancy diminishes for higher arrival rates, as indicated by the downward trend of the curve. We choose this curve as a reference curve for all other algorithms since it can provide a basis for comparing other algorithms in terms of their effectiveness for "smoothing" load over the network. The curves for all other algorithms represent variations from the reference curve when load balancing is employed. These variations are evaluated with respect to the average load. For a small load, these magnitudes are significantly large. This is true for the *PLS*, *FLS*, *PBS* and *FBS* algorithms as shown in Figure 4. The situation changes as the average load increases because the fluctuations in load become small and tasks settle at lightly loaded nodes making the load more balanced.

5.3 Information Collection Overhead

Information exchange incurs overhead due to storage, processing of this information by the Information Collector/Dispatcher and the messages generated for this purpose. This overhead is a function of number of messages generated and can be estimated by the average number of queries per task. Our objective is not to actually assess this overhead; rather we compare this overhead for periodic and fresh load update strategies. In Figure 5, average number of queries per unit-time per node are plotted for *FBS*, *FRS* and *FLS*. The average numbers of queries made by fresh load update algorithms are compared with periodic load update pattern when nodes update their Link Load Views after every 0.2 time units and thus making 5 queries per unit-time. The proposed algorithms *FBS* and *FRS* again show better performance by inducing less overhead as compared to *FLS*. Figure 5 also reveals the advantage of periodic algorithms at high load and the advantage of fresh update algorithms at low load. Periodic algorithms have advantage if T_u is adjusted according to load, ranging from 1 time unit at low load to 0.125 time units at high load.

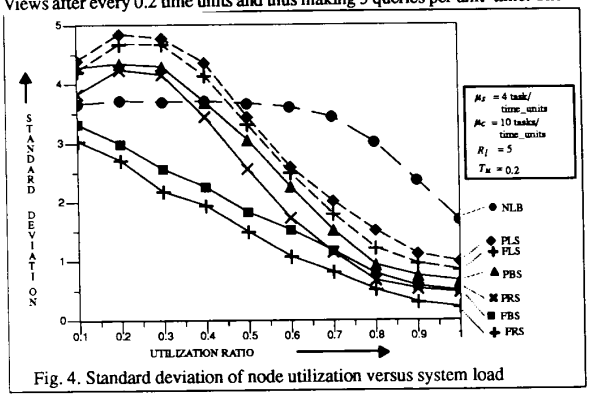

Fig. 4. Standard deviation of node utilization versus system load

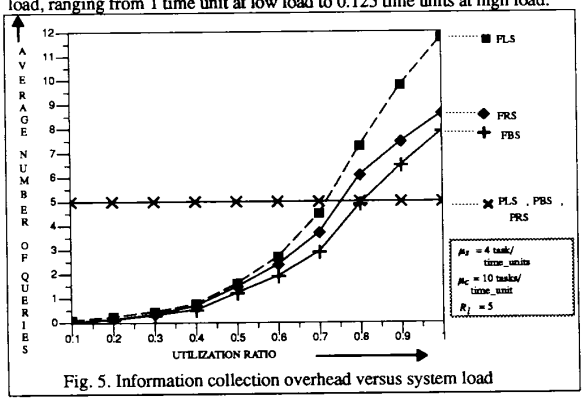

Fig. 5. Information collection overhead versus system load

6 SENSITIVITY STUDY

The results presented in the following sections indicate the sensitivity of each algorithm to different parameters.

6.1 Effect of Communication Rate

Curves of response times versus communication rates at low load and high load are shown in Figure 7 (a) and Figure 7 (b), respectively. The response time decreases almost linearly with increased communication rate. Figure 7 (a), shows that for low communication rate, the proposed algorithms perform better than *FLS* and *PLS* algorithms. At extremely low communication rate, the response time of *PLS* is even poorer than the no load balancing scheme, whereas the proposed algorithms still yield a better response time. As the communication rate starts increasing, a rapid improvement in performance is noticed in all the algorithms.

For communication rate ranging from 4 to 10 tasks/unit-time, the response time decreases rapidly. By increasing the communication, we note that the response time starts decreasing at a slower pace. Very little improvement results when this ratio is increased beyond 5 corresponding to a communication rate of 20 tasks/unit-time. Therefore it does not payoff if the communication rate is increased beyond that. The effect of communication rate on response time at high load is shown in Figure 7 (b). We observe that the proposed algorithms outperform *FLS* and *PLS* algorithms. The difference in performance of the proposed algorithms and those of the *FLS* and *PLS* algorithms are even high at low communication rate with *PLS* and *FLS*

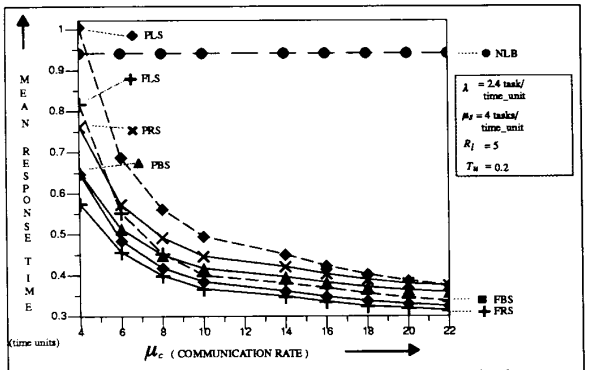

Fig. 7(a).Effect of communication rate on response time at low load

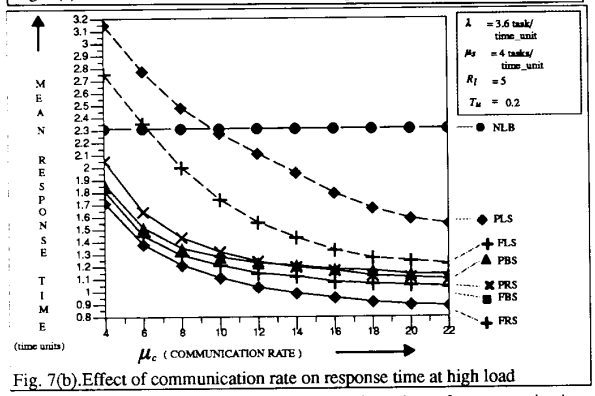

Fig. 7(b).Effect of communication rate on response time at high load

performing even worse than the no load balancing scheme for communication rates below 10 tasks/unit-time and 6 tasks/unit-time, respectively.

Besides reducing communication delay in task migration, faster communication rate reduces contention at the channels. As a result queuing delays at send queues decrease. With a slow network channel, a task may no longer find the destination node a suitable choice as the state of the node may have totally changed when the task finally reaches that node. Generally, all scheduling algorithms face that problem. Once a task reaches a wrong node, that node re-schedules that task resulting in an excessive migration of the task. The better performance of the proposed algorithms, however, can be attributed to the fact that the decision of sending a task to a neighbor is made by tuning the dynamic load threshold. As a result, the scheduling decisions prove more stable as compared to *PLS* and *FLS*.

6.2 Varying Load Status Update Period

As discussed earlier, information collection overhead can be reduced by making periodic updates for high load. For ideal performance, the interval T_u should be tuned according to the system load. But realistically, the arrival pattern can not be predicted forcing us to choose a constant value for T_u. With a

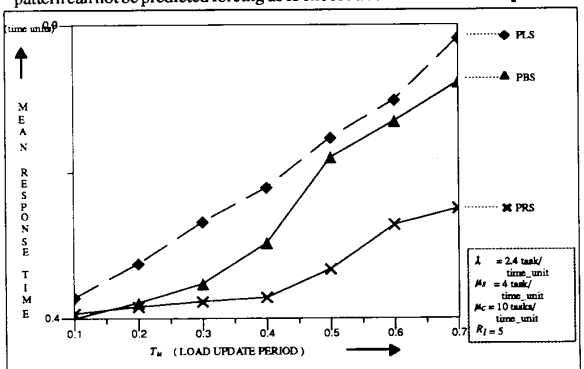

Fig. 8(a). Effect of load update period at low load

small value for T_u, the information is more up to date and the response time can be as good as that of fresh updates, but at the cost of higher overhead. On the contrary choosing T_u to be too long results in wrong scheduling decisions. By analyzing the performance of the proposed periodic algorithms (*PBS* and *PRS*) and comparing them with *PLS*, the impact the accuracy of load status information can be studied. The response time of these algorithms are plotted for various lengths of T_u in Figures 8 (a) and 8 (b) for low and high loads, respectively. The accuracy of information is critical for load balancing since it not only affects the decision whether a task should be serviced locally, but also affects the choice of the target node. Since the rule for the former decision is the same in both the *PBS* and *PRS* algorithms, the difference lies in selection of a target node. In *PRS*, a task is randomly sent to a neighbor not relying on the accuracy of load status information for node selection. In contrast, *PBS* relies more on the accuracy of information because it has to select a least loaded neighbor. This effect can be observed from Figure 8(a). With increasing lose in accuracy of information, the response time of *PBS* becomes worse than the response time for *PRS*. The response time of *PBS* increases rapidly when T_u is increased beyond 0.4 time units. Note that the utilization ratio for this curve is 0.6 corresponding to an inter–task arrival time of 0.42 time units. Since a node is recipient of both externally and internally migrated load, we conjecture that the value of T_u should be less than the inter–arrival time of aggregated task stream The performance of the *PLS* remains inferior to both *PBS* and *PRS* since the decisions made by *PLS* algorithm is more constrained resulting in the response time which increases linearly with T_u. This phenomena is shown in Figure 8 (b). Both algorithms sustain their superiority over *PLS* with better response time even with relatively inaccurate information .

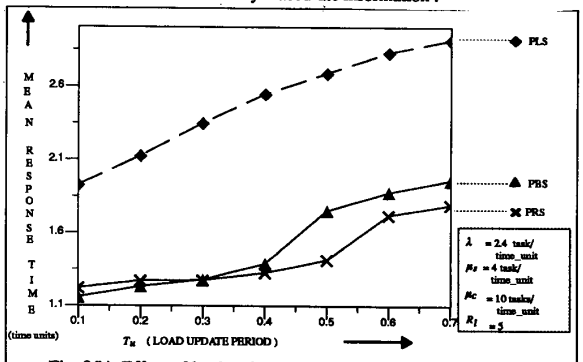

Fig. 8(b). Effect of load update period at high load

6.3 Sensitivity to Transfer Limit

A situation which should be avoided while using load balancing is "task thrashing". Task thrashing occurs when tasks keep on migrating between nodes without settling down for execution at one node. In the proposed schemes, parameter R_l is included to avoid this situation by putting a limit on the maximum number of links a task can traverse. Selecting a high value of R_l provides tasks with more opportunities to find a suitable node. A small value of R_l restricts tasks to only a few migrations possibly preventing tasks from finding a better node. On the other hand, if the tasks are left free to migrate without limit, they may make a large number of unnecessary migrations. Figure 9 (a) shows response times of all six algorithms for different values of R_l. The stability of each algorithm can be observed from Figure 9 (a).

These results can be interpreted as the accuracy of the scheduling decisions made by each algorithm. It can be seen that in *FRS* and *FBS* with one, two or three migrations, tasks can find suitable destinations. For five to six migrations, there is negligible improvement in response time. Increasing the value of R_l beyond eight or nine does not provide any significant impact on performance since tasks settle down before reaching the limit. The curves for *PLS* and *FLS* show signs of some *thrashing* after four or five migrations as the response time starts increasing.Figure 9 (b) shows same results for a utilization ratio of 0.9. The phenomena of *task thrashing* can be easily observed. The response time deceases to a range of minimum values for each algorithm and then it starts increasing. This range varies from algorithm to algorithm. For *PLS* and *FLS*, the range is rather small and the increase in response time is sharp indicating a high degree of thrashing. The second notable observation is how quickly the response time decreases after reaching a peak value. On the other hand, *thrashing in FBS* and *FRS* is minimum. In *PBS* and *PRS*, the tendency of *thrashing* is higher and the response time does not improve even if the transfer limit is increased. For *PLS* and *FLS*, a very high degree of *thrashing* is noticeable which confirms that theses algorithms are less stable. Since R_l is an important parameter for load balancing schemes, these curves also provide a set of values for R_l to prevent *thrashing* and attain the best operating environment for each algorithm. For example, for *FBS* and *FRS*, this range is between 2 and 5 while for *PRS* and *PBS* this range is from 3 to 7.

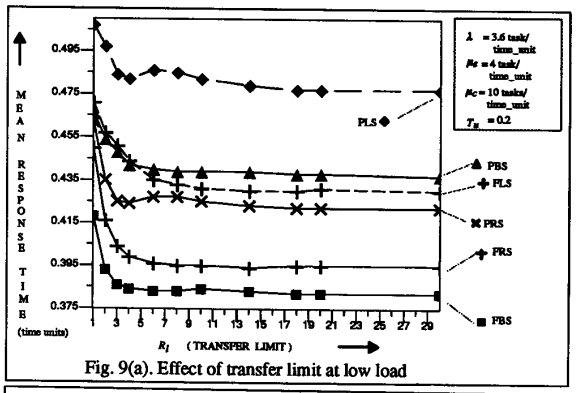

Fig. 9(a). Effect of transfer limit at low load

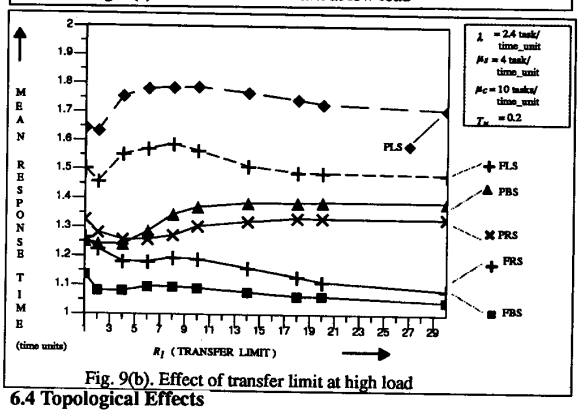

Fig. 9(b). Effect of transfer limit at high load

6.4 Topological Effects

Since the proposed algorithms use information exchange and load balancing confined to neighbors, these algorithms prove suitable to network topologies ranging from a totally symmetric network to a random interconnection structure. The simulation results show that topology has rather little effect on the performance of these algorithms. In addition to the Odd graph shown earlier in Figure 2, we considered ring, chordal ring, fully connected and random topologies, each with 35 nodes. A smaller Odd graph (named as Odd graph 2) with degree 3 was also simulated. The response times obtained for *FBS*, *FRS* and *FLS* for these topologies are given in Table II, for utilization ratio 0.6 and 0.9. The parameters chosen were: μ_s = 4 tasks/unit–time, μ_c =10 tasks/unit–time, R_l = 5 and T_u. To study the variations in the performance of the algorithms, the accumulative average response times and standard deviations are also given in the table. The average response times obtained with the no load balancing scheme are 0.941 and 2.295 for utilization ratios 0.6 and 0.9, respectively.

Overall we can infer from the standard deviation of response times given in these tables that the proposed algorithms are less sensitive to topology compared to *FLS*. At low utilization ratio, such as 0.6, *FBS* proves less sensitive to topology while for high utilization ratios such as 0.9, *FRS* shows the least value of standard deviation showing the least variations. Small value of standard deviation for *FBS* and *FRS*, compared to that of *FLS* shows the adaptive nature of the algorithms. It has already been mentioned that Odd graph gives the best performance and little variations from this value means that the performance does not degrade much for other topologies.

When analyzing the effect of degree, we can consider the two extreme cases of a ring network and a fully connected network. Smaller numbers of links provide a few choices for the task scheduler. Very large number of links gives more choices for node selection but the probability of a wrong decision also increases. From Table II, we observe that the response time of the fully connected network is rather high at utilization ratio 0.6 but becomes the best at the utilization ration of 0.9. This means that a higher degree of connectivity proves more beneficial at high loads. This is due to the fact that if the diameter is small as in the case of a fully connected network (along with a higher degree), the wrong decisions can be rectified more easily. In contrast if the diameter is very large as in the case of a ring, the chances of recovering from wrong decisions are less.

The efficiency of the proposed scheme is obvious from the results obtained on the random network. The response time at utilization ratios 0.6 and 0.9 is only 8 – 9% different from the global average. Comparatively, the standard deviation computed for *FLS* is very high.

6.5 Effect of Variable Number of Nodes

The impact of the number of nodes in the network is also evaluated. Again we only consider fresh load update algorithms *FRS* and *FBS* and compare their performance with *FLS*. The nodes were connected in the form of a ring with the number of nodes varied from 10 to 100. The value of R_l is adjusted for each configuration; it is set equal to the diameter of the ring. The curves for low load (utilization ratio = 0.6) and high load (utilization ratio = 0.9) are shown in figure 10 (a) and 10 (b). In contrast to earlier studies [17] we observe that the

TABLE II
RESPONSE TIME AND COMMUNICATION OVERHEAD FOR VARIOUS TOPOLOGIES

TOPOLOGY	UTILIZATION RATIO = 0.6			UTILIZATION RATIO = 0.9		
	FBS	FRS	FLS	FBS	FRS	FLS
ODD GRAPH	0.457	0.491	0.493	1.092	1.184	1.739
ODD GRAPH-2	0.467	0.499	0.522	1.145	1.222	1.433
RING	0.477	0.580	0.629	1.209	1.279	1.621
CHORDAL RING	0.517	0.557	0.617	1.225	1.296	1.934
RANDOM	0.536	0.613	0.650	1.269	1.323	1.887
FULL CONNEC.	0.539	0.439	0.663	1.041	1.198	1.442
AVERAGE	0.498	0.530	0.596	1.163	1.250	1.676
STANDARD DEV.	0.033	0.059	0.064	0.079	0.051	0.196

aggregated performance of scheduling algorithms does not saturate for moderate network size (such as 35 nodes). Instead, the proposed scheme exhibits a sustained performance as the size of the network increases even up to 100 nodes.

We expect that the saturation will occur for a large number of nodes such as 200 to 300. Referring to Figures 10 (a) and 10 (b), we observe that the response times almost linearly decrease with an increase in the number of nodes for both values of utilization ratios. At low load, both *FBS* and *FRS* maintain their enhanced performance over *FLS*. At high load, the difference in performance becomes significantly large when the number of nodes are varied between 10 to 25. For these curves the transfer limit R_l was adjusted as N – 1. By keeping a high value of R_l , the load can diffuse from one part of the network to another.

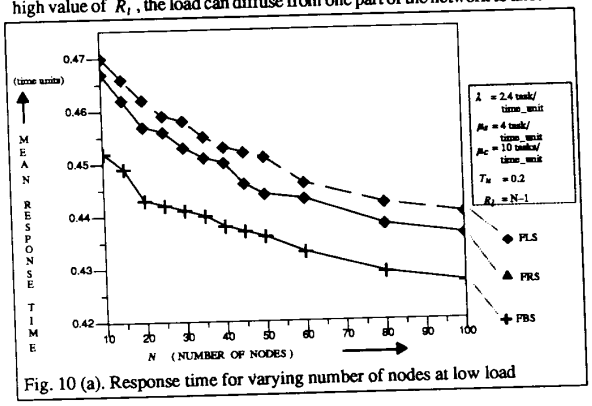

Fig. 10 (a). Response time for varying number of nodes at low load

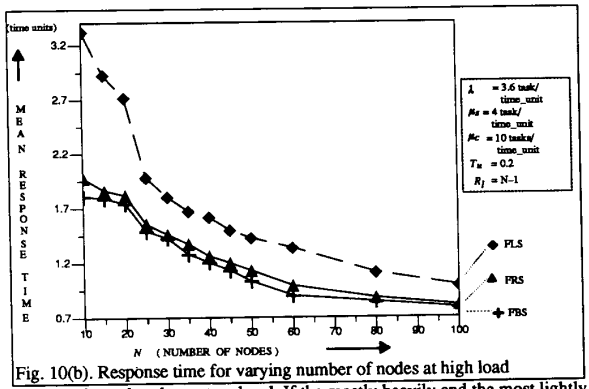

Fig. 10(b). Response time for varying number of nodes at high load

7 Conclusions and Future Work

In this paper we have presented an approach to dynamic task scheduling and load balancing. An extensive study of our approach under a wide range of parameters indicates that this approach is applicable to a more practical environment. The scheduling algorithm supported by some software components improves the response time to a great extent by trying to keep the network nodes equally busy. The exchange of load status information has low complexity. An interesting trade–off between periodic update to non–periodic information update is exhibited. The performance of the proposed algorithms does not degrade when no transfer limit is used. The communication rate of the network has a major influence on the performance but we showed that further reduction in response time can not be achieved after a certain increase in communication rate. The simulation experiments on various network topologies showed the adaptive nature of our algorithms. Even on an irregularly connected network with different number of links per node, the algorithms showed very little tendency of making uneven distribution of load.

Although decentralized load balancing algorithms have gained great popularity, new solutions need to be explored for very large system consisting of hundreds and thousands of nodes. Fully distributed algorithms use small amount of information about the state of the system which makes the scope of scheduling decision very limited. Since gathering large amount of state information may decrease the accuracy, it becomes more appropriate to collect small amount of more accurate information. Small systems can yield good performance with limited information but this may not be true for large systems. Furthermore, if each node takes autonomous decision, load can not be fully balanced at the system level. If the mostly heavily and the most lightly loaded sections of a large network (with a diameter of the order of logarithm of the number of processors) are a large distance apart, a fully distributed algorithm, with limited amount of information may have to executed a number of times to balance the load among these sections. Despite the fact that fully distributed algorithms incur less overhead due to message exchange, this overhead linearly increases with the system size. This may result in a proportional increase in the average response time. On the other hand, centralized algorithms do have the potential of yielding optimal performance, but with a large system, the global information collection becomes a formidable task. The storage requirement for maintaining this information also becomes prohibitively high. Also, such a system is less fault tolerant.

In another study [1], we propose a new approach, which is semi–distributed in nature, for large systems consisting of hundreds of nodes. In that scheme, we introduce the notion of sphere of locality where each sphere, with a central control, is a cluster of nodes. A scheduling algorithm is proposed that is executed by only a set of nodes, called schedulers which are responsible for scheduling tasks within their own spheres. A partitioning strategy, based on a combinatorial structure known as Hadamard Matrix, is suggested for Hypercube and Bisectional graphs. The partitioning strategy determines an optimal number of schedulers. The number of schedulers needs to small in order to have a low overhead. At the same time the schedulers need to be sufficiently enough to effectively manage load within their spheres.

References

[1] I. Ahmad and Arif Ghafoor, "Semi Distributed Scheduling Algorithms for Large Distributed Systems," Technical Report no. TR 90–3, Department of Electrical and Computer Engineering, Syracuse University.

[2] Amnon Barak and Amnon Shiloh,"A Distributed Load balancing Policy for a Multicomputer," *Software Practice and Experience*, vol. 15(9), pp. 901–913, Sept. 1985.

[3] S. H. Bokhari," On the Mapping Problem,"*IEEE Trans. on Computers*, vol. SE–5, no. 5, pp. 207–214, March 1981.

[4] Thomos L. Casavant and John G. Kuhl,"Analysis of Three Dynamic Distributed Load–Balancing Strategies with Varying Global Information Requirements,"*7–th IEEE Intl. Conf. on Distributed Computing Systems*, pp. 185–192, 1987.

[5] Derek L. Eager, Edward D. Lazowska and John Zahorjan,"Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. on Software Eng.*, vol. SE–12, pp. 662–675, May 1986.

[6] Ahmed K. Ezzat, R. Daniel Bergeron and John L. Pokoski."Task Allocation Heuristics for Distributed Computing Systems," *Intl. Conf. on Distributed Systems*. pp. 337–346, 1986.

[7] A. Ghafoor, T.R. Bashkow, I. Ghafoor, "Bisectional Fault–tolerant Communication Architecture for Supercomputer Systems," *IEEE Trans. on Computers*, vol 38, no. 10, pp 1425–1446, Oct 1989.

[8] Anna Ha'c and Xiaowei Jin,"Dynamic Load Balancing in Distributed Decentralized Algorithm," 7–th *IEEE Intl. Conf. on Distributed Computing Systems*, pp. 170–178, 1987.

[9] M. Ashraf Iqbal, Joel H. Saltz and Shahid H. Bokhari,"A Comparative Analysis of Static and Dynamic Load Balancing Strategies,"*Proc. of 1986 Intl. Conf. on Parallel Processing*, pp. 1040–1047, Aug. 1986.

[10] Frank C. H. Lin and Robert M. Keller, "Gradient Model: A Demand Driven Load Balancing Scheme", *Proc. of 1986 Intl. Conf. on Distributed Computing Systems*.

[11] Lionel M. Ni and Kai Hwang,"Optimal Load Balancing in a Multiple Processor System with Many Job Classes," *IEEE Trans. on Software Eng.*, vol. SE–11, pp. 491–496, May 1985.

[12] Lionel M. NI, Chong–Wei Xu and Thomos B. Gendreau,"A Distributed Drafting Algorithm for Load Balancing," *IEEE Trans. on Software Eng.*, pp. 1153–1161, Oct. 1985

[13] Amit P. Sheth, Anoop Singhal and Ming T. Liu,"An Analysis of the Effect of Network Parameters on the Performance of Distributed Database Systems," *IEEE Trans. on Software Eng.*, 1174–1184 Oct. 1985.

[15] John A. Stankovic,"An Application of Bayesian Decision Theory to Decentralized Control of Job Scheduling," *IEEE Trans. on Computers*, pp. 117–129, Feb. 1985.

[16] Alexander Thomasian,"A Performance Study of Dynamic Load Balancing in Distributed Systems," 7–th *IEEE Intl. Conf. on Distributed Computing Systems*, pp. 178–184, 1987.

[17] Songnian Zhou,"A Trace Driven Simulation Study of Dynamic Load Balancing," *IEEE Trans. on Software Eng.*, Vol. 14 No. 9, Sept. 1988.