A DISTRIBUTED SCHEDULING ALGORITHM FOR HETEROGENEOUS REAL-TIME SYSTEMS

Osman ZeinElDine Mohamed El-Toweissy Ravi Mukkamala
Department of Computer Science, Old Dominion University
Norfolk, Virginia 23529.

Abstract

Much of the previous work on load balancing and scheduling in distributed environments was concerned with homogeneous systems and homogeneous loads. Several of the results indicate that random policies are as effective as other more complex load allocation policies. In this paper, we investigate the effects of heterogeneity on scheduling algorithms for hard-real time systems. We propose a distributed scheduler specifically to handle heterogeneities in both nodes and node traffic. The performance of this algorithm is measured in terms of the percentage of jobs discarded. While a random task allocation is very sensitive to heterogeneities, our algorithm is shown to be robust to such non-uniformities in system components and load.

1. Introduction

Meeting deadline is of utmost importance for jobs in hard real-time systems. In these systems, when a job cannot be executed to completion prior to its deadline, it is either considered as having a zero value or as a discarded job with possible disastrous side-effects. Scheduling algorithms are assigned the responsibility of deriving job schedules so as to maximize the number of jobs that meet their deadline. Most of the literature in real-time systems deals with periodic deterministic tasks which may be prescheduled and executed cyclically [1,2,5,6]. The aperiodically arriving random tasks with deadlines have not been thoroughly investigated [3]. In addition, much of the studies in this area assume systems dedicated to real-time applications and working at extremely small loads (10% or less).

In a distributed real-time system, jobs with deadlines are received at each of the nodes in the system. Each node is generally capable of executing a job completely. These jobs are scheduled for execution at the nodes by a scheduler. A distributed scheduler is a distributed algorithm with cooperating agents at each node. Basically, the agents cooperate through exchange of local load information. The decision for scheduling a job, however, is taken by a local agent. Much of the current work in distributed scheduling assumes identical scheduling algorithms, homogeneous processing capabilities, and identical request arrival patterns at all nodes across the distributed system [7].

In this paper, we are interested in investigating the effects of heterogeneity and aperiodicity in distributed real-time systems on the performance of the overall system. We have designed a distributed scheduler specifically aimed at tolerating heterogeneities

in distributed systems. Our algorithm is based on a tree-structured scheduler where the leaves of the tree represent the processing nodes of the distributed system, and the intermediate nodes represent controlling or server nodes. The server node is a guardian for nodes below it in the tree. The leaf nodes attempt to keep its guardian node informed of their load status. When a leaf node cannot meet the deadline of an arriving job, it transfers the job to its guardian (or server). The guardian then sends this job either to one of its other child nodes or to its guardian. We measure the performance of our algorithm in terms of percentage of discarded jobs. (A job may be discarded either by a leaf node or by one of the servers.) Since random scheduling is often hailed to be as effective as some other algorithms with more intelligence, we compare the performance of our algorithm with a random scheduler [4]. Even though a random algorithm is often effective in a homogeneous environment, our investigations found it to be unsuitable for heterogeneous environments.

This paper is organized as follows. Section 2 presents the model of the system adopted in this paper. Section 3 describes the proposed scheduling algorithm. Section 4 summarizes the results obtained from simulations of our algorithm and a random scheduler algorithm. Finally, Section 5 presents some conclusions from this study and proposes some future work.

2. The System Model

For the purposes of scheduling, the distributed system is modeled as a tree of nodes and is shown in Figure 1. (The choice of the hierarchical structure is influenced by our scheduling algorithm which can handle a system with hundreds of nodes. The choice of three levels in this paper is for ease of illustration.) The nodes at the lowest level (level 0) are the *processing* nodes while the nodes at the higher levels represent guardians or *servers*. A processing node is responsible for executing arriving jobs when they meet some specified criteria. The processing nodes are grouped into *clusters*, and each cluster is assigned a unique server. When a server receives a job, it tries to either redirect that job to a processing node within its cluster or to its guardian. It is to be noted that this hierarchical structure could be logical (i.e., some of the processing nodes may themselves assume the role of the servers).

In summary, there are four components in the system model: jobs, processing nodes, servers, and the communication subsystem. A job is characterized by its arrival time, execution time, deadline, and priority (if any). The specifications of a processing node include its speed factor, scheduling policy, external arrival rate (of jobs), and job mix (due to heterogeneity). A server is modeled by its speed and its node assignment policy. Finally, the communication subsystem is represented by the speeds of transmission and distances between different nodes (processing and servers) in the distributed system.

3. Proposed Scheduling Algorithm

We describe the algorithm in terms of the participation of the three major components: the processing node, the sever at level 1, and the server at level 2. The major execution steps involved at these three components are summarized in Figure 2.

3.1 General

First let us consider the actions at the processing node. When a job arrives (either from an external user or from the server) at a *processing node*, it is tested at the gateway. The local scheduling algorithm at the node decides whether or not to execute this job locally. This decision is based on pending jobs in the local queue (which are already guaranteed to be executed within their deadlines), the requirements of the new job, and the scheduling policies (e.g., FCFS, SJF, SDF, SSF etc. [8]). In case the local scheduler decides to execute it locally, it will insert it in the local processing queue, and thereby guaranteeing to execute the new job within its deadline. By definition, no other jobs already in the local queue will miss their deadlines after the new addition. In case the local scheduler cannot make a guarantee to the new job, it either sends the job to its server (if there is a possibility of completion), or discard the job (if there is no such chance of completion).

Let us now consider the actions at level-1 server. Upon arriving at a *server*, a job enters the server queue. First, the server attempts to choose a candidate processing node (within its cluster) that is most likely to meet the deadline of this job. This decision is based on the latest information provided by the processing node to the server regarding its status. This information includes the scheduling algorithm, current load and other status information at each of the processing nodes in its cluster. (The choice of the information content as well as its currency are critical for efficient scheduling. For lack of space, we omit this discussion here.) When more than one candidate node is available, a random selection is carried out among these nodes. (We found a substantial difference in performance between choosing the first candidate and the random selection.) If a server cannot find a candidate node for executing the job, it forwards the job to the level-2 server.

The level-2 server (or *top level server*) maintains information about all level-1 servers. Each server sends an abstract form of its information to the level-2 server. Once again, the information content as well as its currency are crucial for the performance of the algorithm. This server redirects an arriving job to one of the level-1 servers. The choice of candidate servers is dependent on the ability of these servers to redirect a job to one of the processing nodes in their cluster to meet the deadline of the job.

The rule for discarding a job is very simple. At any time, a job may be discarded if the processing node or the server at which the job exists finds that if the job is sent elsewhere it would never be executed before its deadline. This may be due to the jobs already in the processing queue, and/or the communication delay for navigation along the hierarchy.

3.2 Information Abstraction at Different Levels

The auxiliary information (about the load status) maintained at a processing node or a server is crucial to the performance of the scheduling algorithm. Besides the information content, its structure and its maintenance will dictate its utility and overhead on the system. To maximize the benefit and minimize the overhead, every level is assigned

its own information structure. The information at the servers is periodically updated by nodes at the lower level. (The time interval for propagating the information to the servers is a parameter of the system.)

The jobs waiting to be executed at a processing node are classified according to the *local* scheduling algorithm (e.g., the classification would be on based priority if a priority scheduler is used.). Due to this dependency on the scheduling algorithm, we allow each processing node to choose its own local classification. Typically, the following attributes are maintained for each class:

- the average response time; (used for performance statistics)
- the likely end of execution (time) of the last job, including the one currently being served;
- the minimum slack among the jobs currently in the processing queue.

In addition, depending on the scheduling policy, we may have some other attributes.

The server maintains a copy of the information available at each of its child nodes including the scheduling policy. Since the information at a child node is dependent on the local scheduling policy, the server node should be capable of maintaining different types of data. Using this information, the server should be able to decide which processing nodes are eligible for executing a job and meet its deadline.

The information at level 2 server consists of an abstraction of the information available at each of the level-1 servers. Since each level-1 server may contain nodes with heterogeneous scheduling policies, level-2 server abstracts its information based on scheduling policies for each server. Thus, for a FCFS scheduling policy, it will contain abstracted status information from each of its child servers. This is repeated for other scheduling policies. Thus, the information at this level does not represent information at a processing node; rather it is a summary of information of a group of nodes in a cluster with the same scheduling policy.

4. Results

In order to evaluate the effectiveness of our scheduling algorithm, we have built a simulator and made a number of runs. The results presented in this paper concentrate on the sensitivity of our algorithm to four different parameters: the cluster size, the communication delay (between nodes), the frequency of propagation of load information (between levels), and the node heterogeneity. For lack of space, we have omitted other results such as the scheduler's sensitivity to heterogeneity in scheduling algorithms, heterogeneity in loads, and the effects of information structures. Accordingly, all the results reported here assume:

- FCFS scheduling policy at every node,
- the total number of processing nodes is 100,
- equal load at all nodes,
- communication delay between any nodes is the same.

The performance of the scheduler is measured in terms of the percentage of jobs discarded by the algorithm (at levels 0,1, 2). The rate of arrivals of jobs and their processing requirements are combinedly represented through a load factor. This load factor refers to the load on the overall system. Our load consists of jobs from three types of execution time constraints. The first type are short jobs with average execution of 10 units of time and with a slack of 25 units. (The actual values for a job are derived from an exponential distribution.) The second type of jobs have an average execution time of 50 units and a slack of 35 units. Long jobs have average execution times of 100 units and slacks of 300 units. In all our experiments, all these types have equal contribution to the overall load factor.

We now discuss our observations regarding the characteristics of the distributed scheduler in terms of the four selected parameters. In order to isolate the effect of one factor from others, the choice of parameters is made judiciously. For example, in studying the effects of cluster size (Figure 3), the updation period is chosen to be a medium value of 200 (stat=200), the communication delay is chosen to be small (com=5), and the nodes are assumed to be homogeneous (node: hom). Similarly, while studying the effects of the updation period (Figure 4), the cluster size is chosen to be 100 (cluster =100). Similar choices are made in the study of other two factors.

4.1 Effect of Cluster Size

Cluster size indicates the number of processing nodes being assigned to a level-1 server. In our study, we have considered three cluster sizes: 100, 50, and 10. A cluster of 100 nodes indicates a centralized server structure where all the processing nodes are under one level-1 server. In this case, level-2 server is absent. Similarly, in the case of cluster of 50 nodes, there are two level-1 servers, and one level-2 server. For 10-node cluster, we have 10 level-1 servers. In addition, we consider a completely decentralized case represented by the *random* policy. In this case, each processing node acts as its own server and randomly selects a destination node to execute a job which it cannot locally guarantee. We make the following observations (Figure 3).

- Both cluster sizes of 100 and 50 nodes have identical effect on performance.
- With cluster size of 10, the percentage of discarded jobs has increased. This difference is apparent at high loads.
- The random policy results in a significantly higher percentage of jobs being discarded.

From here, we conclude that our algorithm is robust to variations in cluster size. In addition, its performance is significantly superior to a random policy.

4.2 Effect of The Frequency of Updations

As is the case for all distributed algorithms, the currency of information at a node about the rest of the system plays a major role in performance. Hence, if the state of processing nodes vary rapidly, then the frequency of status information exchanges

between the levels should also be high. In order to determine the sensitivity of the proposed algorithm to the period of updating statistics at the servers, we experimented with four time periods: 25, 100, 200 and 500 units. (These time units are the same as the execution time units of jobs.) The results are summarized in Figure 4. From here we make the following observations.

- Our algorithm is extremely sensitive to changes in period of information exchanges between servers and processing nodes.
- Even in the worst case of 500 units, the performance of our algorithm is significantly better than the random policy.

4.3 Effect of Communication Delay

Since processing nodes send jobs that cannot be executed locally to a server, communication delay is a major factor in reducing the number of jobs discarded due to deadline limitations. Here, we present results for four values of of communication delay: 0, 5, 10 and 20 units. (These units are the same as the execution time units of jobs.) The results are summarized in Figure 5. The communication delay of zero represents a closely coupled system with insignificant time of inter-node or inter-process communication. A higher communication delay implies lower slack for jobs that cannot be executed locally. It may be observed that

- When the communication delay is 0, 5, or 10, the number of discarded jobs with our algorithm (DSA) is relatively small and independent of this delay. In all these cases, the percentage of discarded jobs with DSA is much smaller than with random policy.
- When communication delay is 20, however, the percentage of discarded jobs is much higher. In fact, in this case the random policy has a better performance than our algorithm.
- The performance difference between random policy and our algorithm reduces with the increase in communication delay. When the communication delay is higher, this difference has vanished, and in fact the random policy has displayed better performance.

We attribute our observation to the selection of slacks for the input jobs. If a shortest job could not be executed at the processing node at which it originated, it has to go through two hops of communication (node to server, server to node), resulting in twice the delay for a single hop. Since the maximum value of the slack for jobs with short execution time has been taken to be 25 units of time (in our runs), any communication delay above 12 units will result in a non-local job being discarded with certainty. Thus, even though our algorithm is robust to variations in communication delay, there is an inherent relationship between the slack of an incoming job and the communication delay.

4.4 Effect of Node Heterogeneity

As mentioned in the introduction, a number of studies claim that sending a job randomly over the network would be almost as good as using a complex load balancing algorithm [4]. We conjecture that this claim is only valid under homogeneous nodes assumption and jobs with no time constraints. Since one of our major objectives has been to test this claim for jobs with time constraints over a set of heterogeneous nodes, experiments have been conducted under four conditions. For each of these conditions, we derive results for our algorithm (DSA) as well as the random policy. The results are summarized in Figure 6. The homogeneous case (denoted by *hom*) represents a system with all 100 nodes having the same unit speeds. (Since a job is only distinguished by its processing time requirements, we have considered only speed heterogeneities.) The three heterogeneous cases are represented by *het1, het2, het3*. The heterogeneities are described through a set of <# of nodes, speed factor> pairs. For example, *het1* relates to a system with 50 nodes with a speed factor of 0.5 and 50 nodes with a speed factor of 1.5. Thus the average speed of a processing node is still 1.0, which is the same as the homogeneous case. The other two heterogeneous cases may be similarly explained. Among the cases considered, the degree of heterogeneity is maximum for *het3*. From our results it may be observed that:

- With our algorithm, even though the increase in degree of heterogeneity resulted in an increase of discarded jobs, the increase is not so significant. Hence, our algorithm appears to be robust to node heterogeneities.
- The performance of the random policy is extremely sensitive to the node heterogeneity. As the heterogeneity is increased, the number of discarded jobs is also significantly increased.

With the increase in node heterogeneity, the number of nodes with slow speed also increase. Thus, using a random policy, if a slow speed node is selected randomly, then the job is more likely to be discarded. In our algorithm, since the server is aware of the heterogeneities, it can suitably avoid a low speed node when necessary. Even in this case, there is a tendency for high-speed nodes to be overloaded and low speed nodes to be under loaded. Hence, the difference in performance.

5. Conclusions

In this paper, a distributed algorithm has been proposed to help schedule jobs with time constraints over a network of heterogeneous nodes, each of which could have its own processing speed and job-scheduling policy. Several parametric studies have been conducted. From the results obtained it can be concluded that:

- the algorithm has a large improvement over the random selection in terms of the percentage of discarded jobs;
- the performance of the algorithm tends to be invariant with respect to node-speed heterogeneity;

- the algorithm efficiently utilizes the available information; this is evident from the sensitivity of our algorithm to the periodicity of update information.
- the performance of the algorithm is robust to variations in in the cluster size.

In summary, our algorithm is robust to several heterogeneities commonly observed in distributed systems. Our other results (not presented here) also indicate the robustness of this algorithm to heterogeneities in scheduling algorithms at local nodes. In future, we propose to extend this work to investigate the sensitivity of our algorithm to other heterogeneities in distributed systems. We also propose to test its viability in a non-real time system where response time or throughput is the performance measure.

ACKNOWLEDGEMENT

This research was sponsored in part by the NASA Langley Research Center under contracts NAG-1-1114 and NAG-1-1154.

References

[1] S. R. Biyabani, J.A. Stankovic, and K. Ramamritham, "The integration of deadline and criticalness in hard real-time scheduling," *Proc. Real-time Systems Symposium*, pp. 152-160, December 1988.

[2] J.-Y. Chuang and J.W.S. Liu, "Algorithms for scheduling periodic jobs to minimize average error," *Proc. Real-time Systems Symposium*, pp. 142-151, December 1988.

[3] D. W. Craig and C.M. Woodside, "The rejection rate for tasks with random arrivals, deadlines, and preemptive scheduling," *IEEE Trans. Software Engineering*, Vol. 16, No. 10, pp. 1198-1208, Oct. 1990.

[4] D. L. Eager, E.D. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems," *IEEE Trans. Software Engineering*, Vol. SE-12, No. 5, pp. 662-675, May 1986.

[5] R. Rajkumar, L. Sha, and J.P. Lehoczky, "Real-time synchronization protocols for multiprocessors," *Proc. Real-time Systems Symposium*, pp. 259-269, December 1988.

[6] K.G. Shin, C.M. Krishna, and Y.-H. Lee, "Optimal resource control in periodic real-time environments," *Proc. Real-time Systems Symposium*, pp. 33-41, December 1988.

[7] J. Stankovic and K. Ramamritham, "Evaluation of a bidding algorithm for hard real-time distributed systems," *IEEE Trans. Computers*, Vol. C-34, No. 12, pp. 1130-1143, Dec. 1986.

[8] W. Zhao, K. Ramamritham, and J. Stankovic, "Scheduling tasks with resource requirements in hard-real time systems," *IEEE Trans. Software Engineering*, Vol. SE-13, No. 5, pp. 564-577, May 1987.

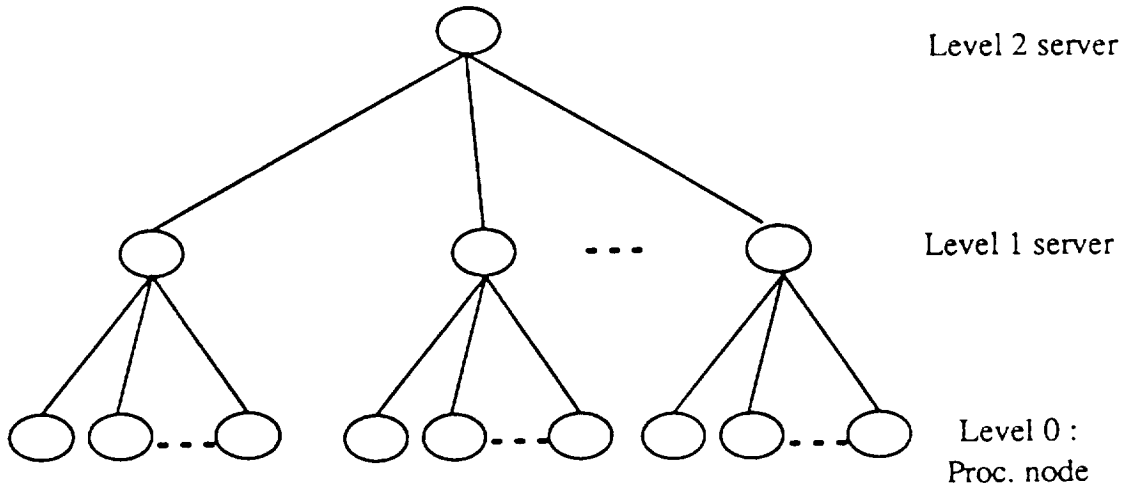


Figure 1 : System Model

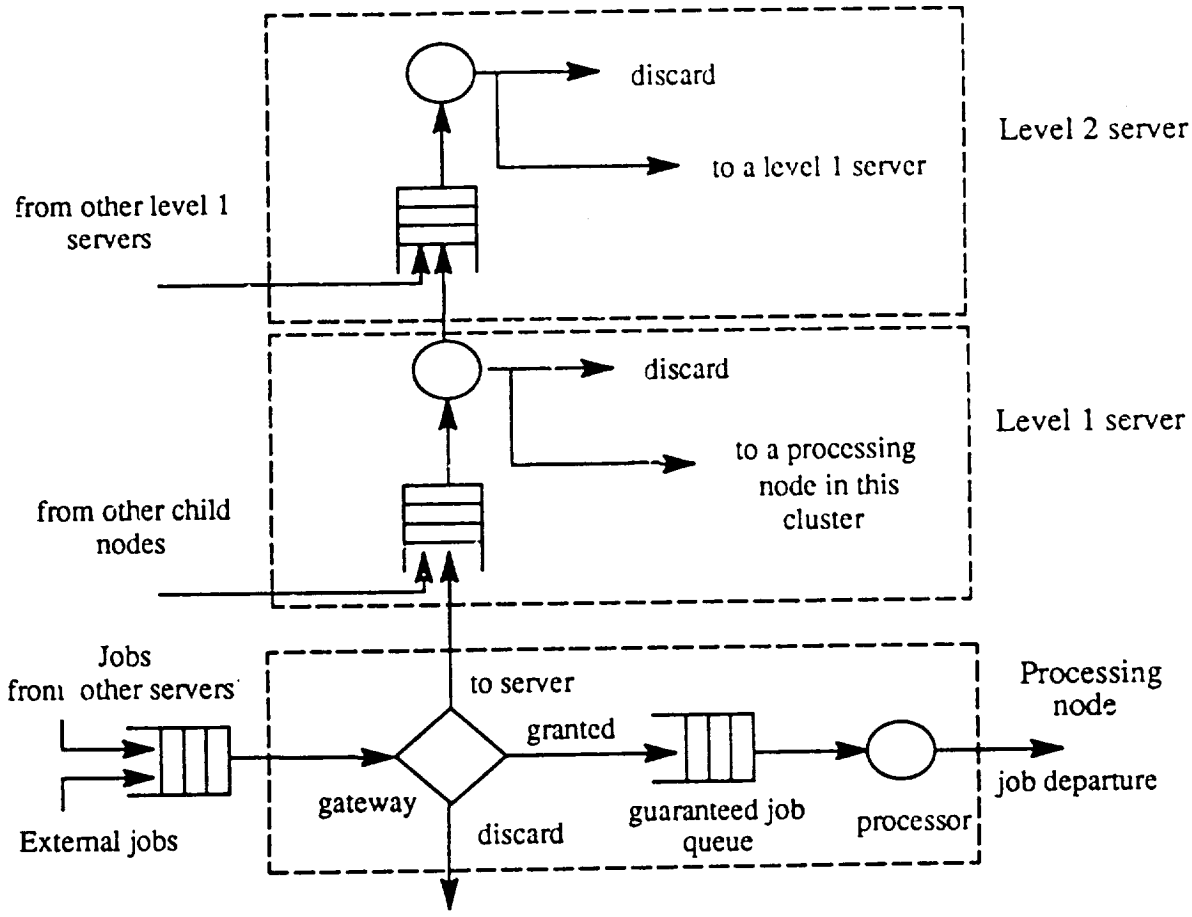


Figure 2 : Flow diagram of The Algorithm

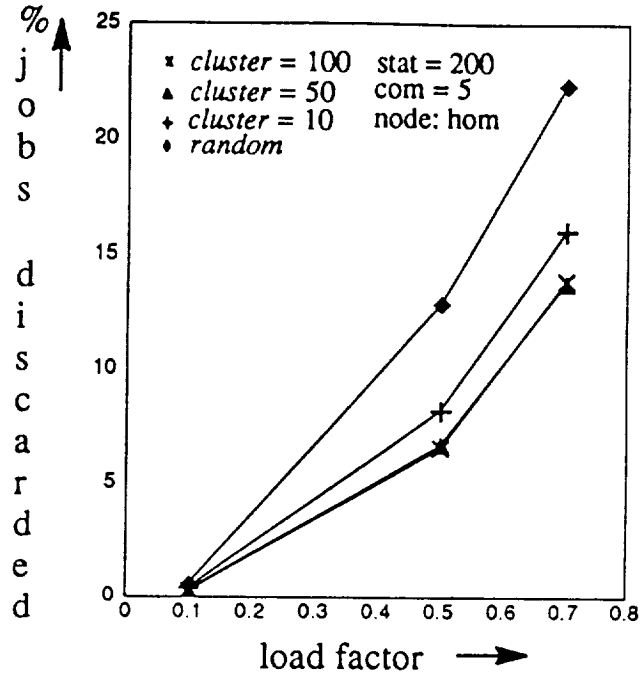


Figure 3:Effect of cluster size

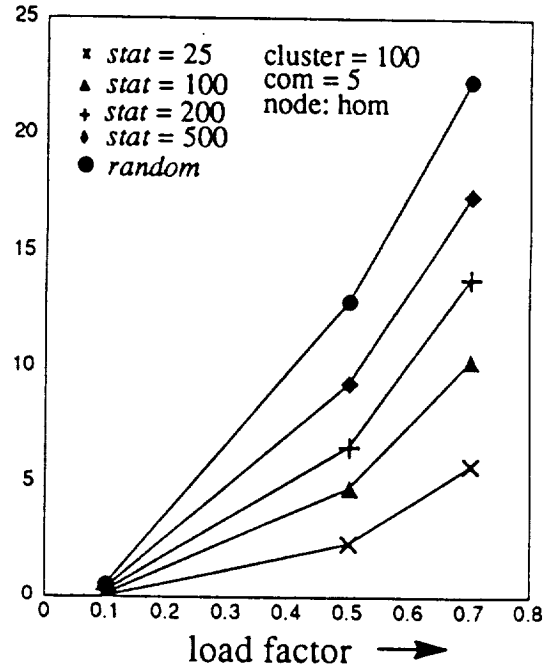
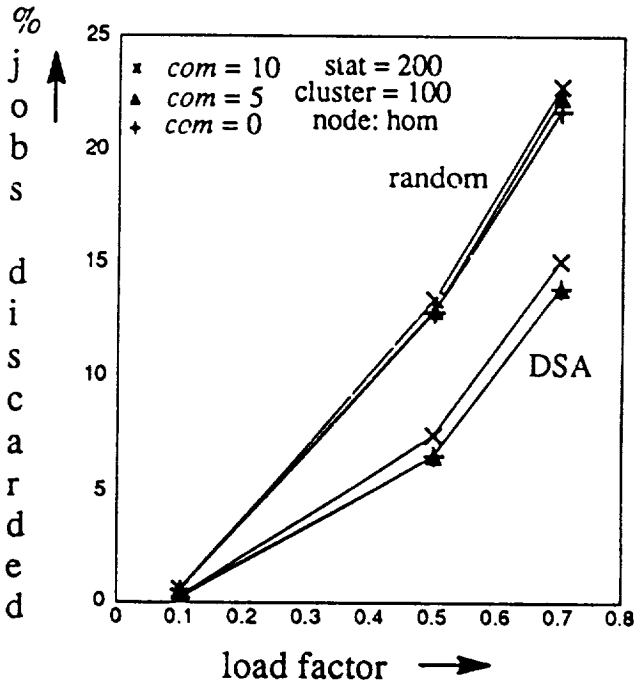


Figure 4: Effect of updation period



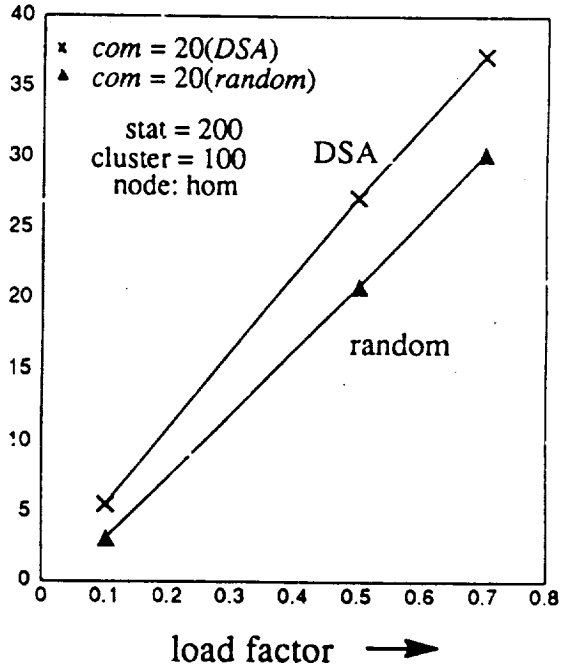


Figure 5:Effect of communication delay

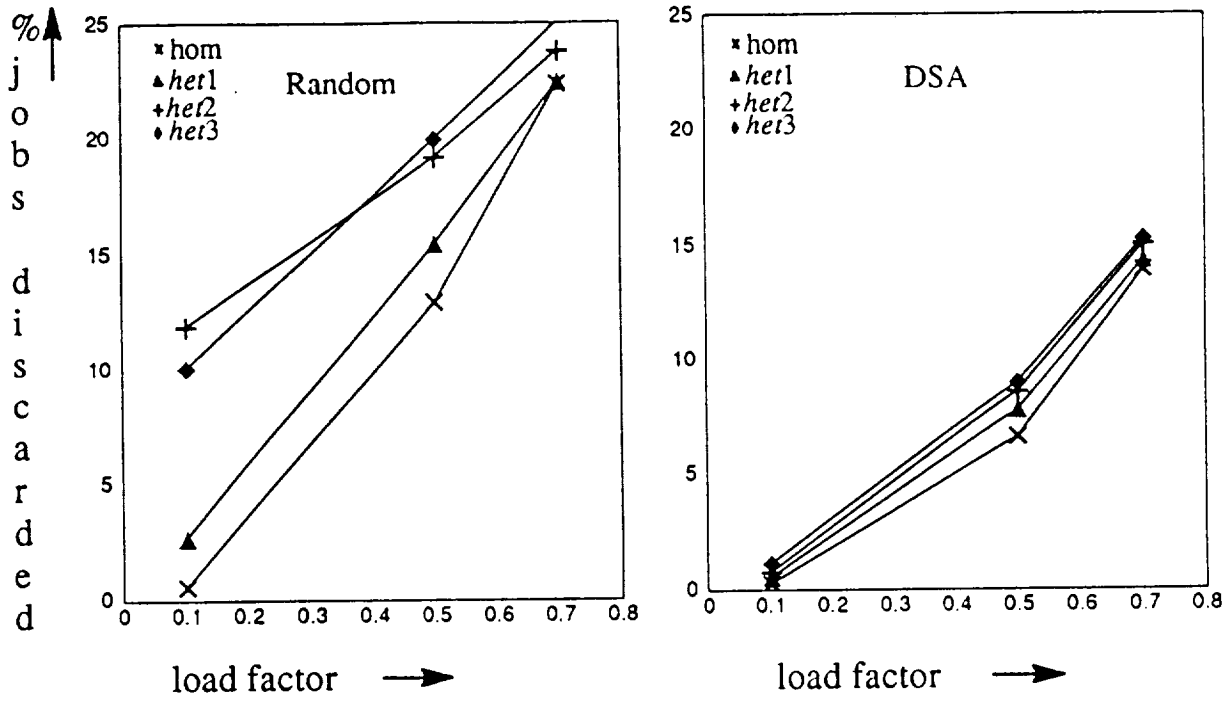


Figure 6: Effect of Node Heterogeneity

het1 : <50,1.0> , <25,1.5> , <25,0.5>
het2 : <50,0.5> , <50,1.5>
het3 : <20,0.25> , <20,0.5> , <20,1.0> , <20,1.5> , <20,1.75>

IEEE PROCEEDINGS OF THE

SOUTHEASTCON '91

NASA

Volume 2

91CH2998-3