01 Jan 1990

Experimental Comparison Of Bidding And Drafting Load Sharing Protocols

Andrew Ross

Bruce M. McMillin
Missouri University of Science and Technology, ff@mst.edu

Recommended Citation

A. Ross and B. M. McMillin, "Experimental Comparison Of Bidding And Drafting Load Sharing Protocols," *Proceedings of the 5th Distributed Memory Computing Conference, DMCC 1990*, vol. 2, pp. 968 - 974, article no. 556306, Institute of Electrical and Electronics Engineers, Jan 1990.
The definitive version is available at https://doi.org/10.1109/DMCC.1990.556306

EXPERIMENTAL COMPARISON OF BIDDING AND DRAFTING LOAD SHARING PROTOCOLS

Andrew Ross and Bruce McMillin‡
Department of Computer Science - University of Missouri-Rolla
Rolla, MO 65401 - ff@cs.umr.edu

ABSTRACT

In recent years, a dramatic rise in the number of personal workstations interconnected via local area networks has occurred in the workplace. These can be organized as distributed computing systems. The combined computing power of these systems are often greater than mainframes of a decade ago, and usually less expensive. There is a growing interest in harnessing this often underutilized power. Researchers are focusing their attention on remote execution of processes as one solution. An additional topic of research is to balance a workload among a series of computers. Remote execution is made possible because the distributed operating system provides migration of a process from one processor to another. The goals of dynamic process migration are to be user transparent, network topology independent, and to achieve a useful compromise between maximum processor utilization and minimum communication overhead. Two load sharing algorithms have been proposed as the "better" solution. The *drafting* algorithm is receiver initiated by a low load processor while the *bidding* algorithm is sender initiated by a high load processor. To date there has been no known published comparison of implemented systems. The purpose of this paper is to compare the performance of each for a real implementation rather than a paper study.

1. INTRODUCTION

A distributed system is formally defined as a collection of autonomous processors that communicate with each other and that allow resource sharing [3]. Currently, locally distributed systems share primarily data, data storage devices, and output devices; there is little sharing of *computational* resources [4]. Transferring some of the workload from highly loaded processors to lightly loaded processors should improve the overall performance of the system.

‡ This work was supported in part by the National Science Foundation under Grant Numbers MIP-8909749 and CDA-8820714, in part by the AMOCO Faculty Development Program, in part by the Manufacturing Research and Training Center (MRTC), and in part by the McDonnell Douglas Corporation.

There are two approaches to the load sharing philosophy. The simple static policy uses predetermined values based on averages to make migration decisions. Adaptive policies are more complicated because they require current system state information to make decisions. Greater performance over a wide workload can be expected with adaptive policies, but care must be taken to ensure the load sharing protocol does not significantly add to the system load. Only adaptive techniques are considered in this paper.

The goals of a dynamic process migration protocol are to be user transparent, network topology independent, and to achieve a useful compromise between maximum processor utilization and minimum communication overhead. Clearly, increasing the amount of remote execution increases communication delay on the network. This adds to the system load. It should be mentioned that no attempt will be made to balance the workload across the system. Instead, the idea is to keep all processors busy and thus reduce the average process response time.

The main obstacles in creating an effective process migration protocol are the estimation of the processor load, expected runtime of a process, and the creation of extra communication traffic by the protocol [2]. Estimating the expected runtime of a process for input to the job scheduler has no guaranteed solution, therefore it was not considered in the design of the algorithms. Many proposals suggest that load determination should be based on numerous measurable parameters. These include resource demands, instruction mixes, number of processes, machine architecture and speed [6]. It has also been suggested in the literature that effective results can be achieved using simple parameters [1][2]. In this paper, the load was determined by the length of a queue of jobs submitted. Although this doesn't measure the actual load of the CPU, observations show this accurately reflects the load of the CPU.

A general description of the bidding and drafting algorithms are given below. Each follows the model of migration protocols described in [1-5]. Specific parameter values chosen will be covered in the design section, and then the results of the comparison are presented. In both algorithms, the protocol runs on each processing node and jobs are submitted to the system at each node.

0-8186-2113-3/90/0000/0968$01.00 © 1990 IEEE

Bidding Algorithm

The main points of the bidding protocol are:

- The load is classified as low (L) or high (H) state.
- when a processor enters a high state it broadcasts a *solicit bids* message to all other nodes.
- all nodes in low state calculate a bid based on its load, and submits it to the requesting processor.
- the high load processor evaluates all bids received and chooses the best node to migrate a job to.
- a *verify* message is sent to the owner of the winning bid to ensure that it can still receive a job.
- if the situation hasn't changed then a job is migrated. If it has, a *too late* message is sent to the initiating processor and the process is restarted.

One well known problem with bidding is that many high load processors may gang up and dump a significant load on the winner of the bids.

Drafting Algorithm

In the drafting protocol, migration is initiated by low load processors in an attempt to improve performance. It is believed to be a better algorithm because bidding may cause many high state nodes to dump onto a single low state node, drafting does not give high state nodes added work, and achieves fairness [2]. The main points are:

- the load is classified as low (L), normal (N) or high (H) state. Normal state nodes will neither accept nor migrate jobs.
- each node maintains a table of load values, the *load table*, of every other node.
- when a processor changes load state it broadcasts a load change message.
- a processor in low load or entering low load checks its load table for the existence of high load processors. If any are discovered it executes a *SendDraftRequest*.
- on receiving a draft request message, a high load machine performs a *RespondDraftRequest*. This entails sending a *draft-age* message to the drafting processor. The draft-age is some measure of the load with respect to draftable processes.
- after the drafting processor receives all the draft age replies, or times out, it will calculate a *draft-standard* based on the draft-ages.
- a *draft-select* message is sent to the chosen high load processor.
- if that node is still in high load it selects and migrates a job. Otherwise a too late message is returned and the low load processor restarts the protocol.

2. IMPLEMENTATIONS

The migration protocols were implemented from common specifications by diverse design groups in the C language on a token ring local area network composed of IBM 6152 processors and a shared file server. Thus, interprocessor communication has a non-negligible delay. Each 6152 is running the UNIX BSD 4.3 operating system and has the same memory capacity and processing speed.

Job Description

A job is a file consisting of 1 to n distinct process names. Each process name is a pointer to an executable program that performs a loop of I/O intensive instructions. There were five jobs of varying runtimes used during the comparison. Each job was first run independently of the migration protocol to determine its standalone runtime (Table 1). During system testing, the runtime of each completed job was recorded and compared to the expected runtime for the system's calculated load.

For example, job file io.1 looks like:

```
3          (* the number of processes  *)
di0002     (* process number 1        *)
di0004     (* process number 2        *)
di0001     (* process number 3        *)
```

and a process would be a simple C program which writes some number of bytes to "/dev/null" to simulate I/O loading.

The following information is associated with each job in the queue:

- machine number where the job originated
- pointer to process within job that is to be started next
- time the job entered the queue.

Jobs are submitted to the processing nodes of the system with rate (λ), however, the pseudo-random sequence is preserved for repeatability at each node. Communication messages include exchange of load estimation, coordination of processes involved in potential process migration, and process migration. The system used in this experiment contains a remote file server which allows multiple processing nodes to have access to the same job. Addresses of jobs and not jobs themselves are migrated. All messages used were 50 bytes or less including message header. This is not very much bandwidth. To transfer an actual job of many thousands of bytes would add congestion to the network and reduce performance. This is important because different results

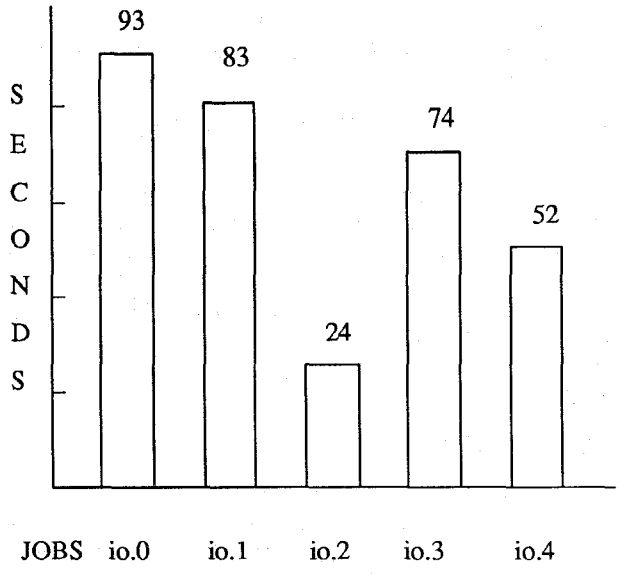

Table 1. Job runtime.

might be obtained on a distributed system not based on a shared file server.

Threshold

The *number-in-queue* is used as the measure of the processor load. The load value is compared to a *threshold* value to determine the high(H)/low(L) or high(H)/normal(N)/low(L) load states. The migration protocol at a particular node is activated based on a change in the load state of either the local or a remote node, depending on the protocol. The threshold is a protocol parameter that is supplied at execution time. This is the cutoff between low and high or low and normal and normal and high loads. It is a relative value based on the runtime of jobs.

Bid Selection

The bidding program was implemented with the owner of the first matching bid being selected as the destination for migration. Results compared to the drafting algorithm indicate that this may be a better decision. In the time it takes drafting to wait for all replies, select the best choice, and respond, the bidding program has already migrated and restarted the job. If there exists prior knowledge of job type, then it might be beneficial to determine the "best" job for migration. An algorithm to ensure that the first response is the "best" response is given in the results section. It could be used by either protocol.

Drafting Design

Where the job originated is used to determine if that job can be migrated, and if it has a high execution priority. A job can only be migrated once to prevent job thrashing [1]. Migrated jobs are also given a higher priority to be started before unmigrated jobs.

The time that the job entered the queue is used in the draft selection process. The draft age reported is chosen to be the waiting time of the oldest job in the queue.

When the migration is triggered by a load table state change and the migration connection has been established, a job is selected and sent to the message passing module for migration. The decision was made to always select the second job in the queue for migration. This was done for its simplicity and fairness to the jobs. An alternative choice is that jobs closer to completion should not be migrated in favor of migrating the youngest job. However, under this design, migrated jobs are given higher priority at the new node so migrating young jobs would be unfair to older jobs at the destination.

3. RESULTS

Data was collected by running pairs of tests for periods of 15 minutes to 2 hours. Each pair consisted of a drafting and bidding migration. The job sequence, rate of job submitting, and high state threshold value were varied to change the expected load on the system. Some testing characteristics were: migration disabled, medium load on all nodes, high load on one processor, and high load on multiple processors.

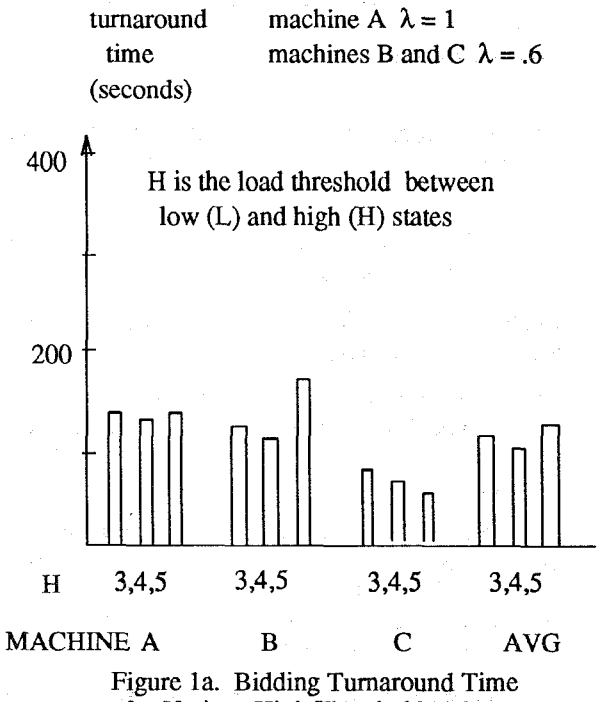

Figure 1a. Bidding Turnaround Time for Various High Threshold Values

Figure 1a illustrates the results of varying the threshold value on job turnaround time for the bidding algorithm. Notice that even though machine A had a higher

job submit rate, machine B had a longer run times for threshold value of 5. This was caused by a high rate of job thrashing which caused machine B to take on high work loads.

The bidding protocol originally outlined in [9] is susceptable to thrashing due to rapid state changes [2]. If a job is submitted to a node which causes the *number-in-queue* to go from 3 (L) to 4 (H) the bidding is activated. If a job were to complete at this time the load would drop from high back to low and the migration process would have to be terminated. This was discovered to happen quite often in our tests compounded by that node getting a new job and entering high load again. One school of thought is that if a node recently entered high load it is more likely to enter high load again. Thus, to make a fair comparison to drafting, a damping factor was introduced. The cutoff from high to low load was taken to be [original_threshold -1]. With the cutoff for high to low being offset by one, the processor has extra time to complete a migration. This damping effect reduced the number of aborted migrations due to the completion of a job.

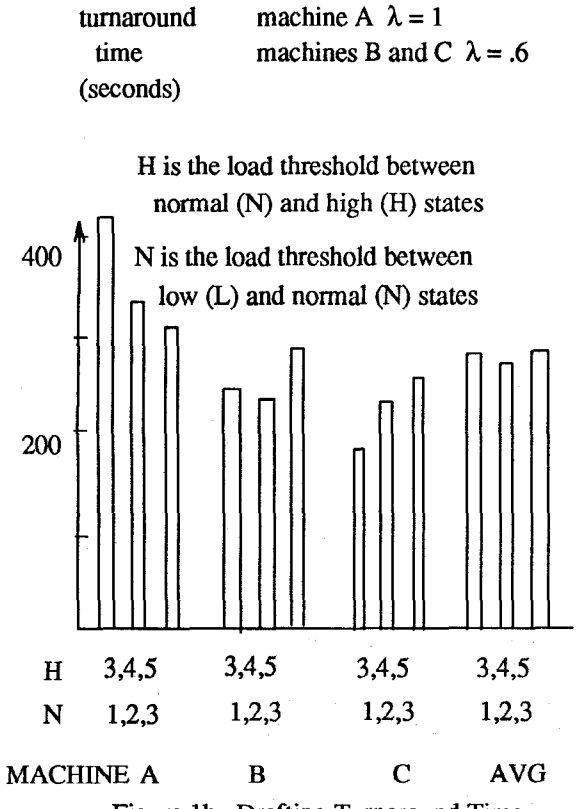

Figure 1b. Drafting Turnaround Time for Various High/Normal Threshold Values

In the drafting implementation, the threshold value between low and normal is chosen to be 2 (L). The threshold value for normal to high is 4 (H), and again the cutoff for high to normal is 3 (N). The threshold damping prevented many aborted migrations as it did for bidding. Figure 1b shows the turnaround time for different threshold values. The graphs of H=5 N=3 show the three processing nodes as being closer to a balanced throughput. This is a good example to point out that balanced does not necessarily mean better. The increased operating system overhead to swap jobs can add to the load. In this case it is better to just keep each node busy.

Is a migration protocol worth implementing? That is one question that these tests hoped to answer. What are the effects of migration on a normal and high load system? And importantly, is one protocol better than the other?

The main measurement for this protocol comparison is the average load induced on the system and the system throughput. The performance of any one job with and without migration was not considered. The average load is inversely related to system throughput. On the system used for implementation, the lower the number of jobs waiting for processor time the faster the turnaround time. In other words, if the same number of jobs are submitted to two systems, then the system with the lower average load has completed more jobs. The number of messages passed and the number of migrations are two system measurements that were not as important in this comparison. The number of migrations was monitored to identify thrashing.

The estimation of load was determined as the length of the job queue. Although this doesn't measure the actual load of the CPU, observations showed this accurately reflects the load of the CPU. In many hours of observation, the load that the drafting program placed on the CPU was less than 0.5. The bidding program was slightly less. Each of the processes that was forked and executed was seen by the system as one process, and so the load increases by 1.0 for each process executed simultaneously. Table 2 compares average job runtime to expected runtime for a system with one job submitted every 60 seconds ($\lambda = 1$).

The expected average was calculated by multiplying the load value by the sum of the completed job's known runtime (table 1). Due to I/O overlap effects, the system ran better than expected for both protocols. This could mean the load measurement is slightly high. Also note that the bidding program produced better results.

	Bidding	Drafting
average system load	4.5	5.6
job runtime average for tests	244.7	353.0
Expected Runtime Average	259.7	361.9

Table 2. System Load and Average Job Runtime for a Sample Test

The first result found was that if the entire system is in a high state, and very little migration is occurring, the system ran slower than with the migration disabled com-

pletely. In Figure 2 the average system load during a bidding test where all nodes had a $\lambda = 1$ is compared to a test with the same system parameters with the migration procedure supressed. When all nodes are in high load state (load > 3) the average system load with migration becomes greater than the average system load without migration. This is due to the overhead of the migration process that does no useful work if the load is balanced (or all processors busy).

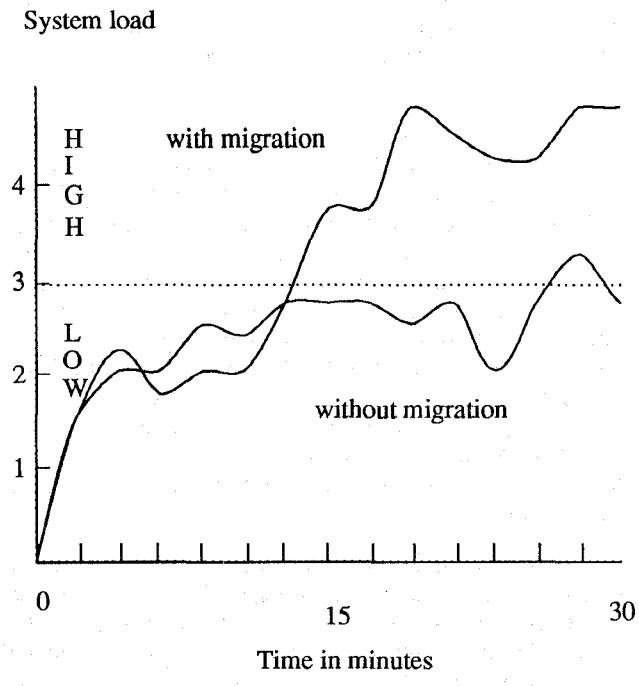

Figure 2. Comparison of Bidding Algorithms With and Without Migration
$\lambda = 1$ for all nodes

The bidding program experienced a problem with state woggling and process thrashing [9] (Figure 3.) as predicted by [2]. State woggling is the term used when a processor frequently changes its state back-and-forth between H-load and N-load or between N-load and L-load. In a 15 minute test with two processes having a high submit rate, $\lambda = 3$, and the rest of the nodes had a low rate of $\lambda = .3$, there were 121 total jobs submitted, and the bidding protocol caused 130 migrations. (Jobs are allowed to migrate more than once in bidding and at most once in drafting.) In particular, 3 jobs migrated 10, 15, 25 times respectively, however, most jobs had zero migrations. Over all bidding tests a 2:1 ratio of jobs submitted to number of migrations was observed.

Even with this thrashing, bidding performed better than drafting. If the actual job were migrated across the network instead of a job pointer, the bidding algorithm would suffer severely under these conditions. The system load for this test was 5.3 for bidding and 6.9 for drafting. Drafting had only 12 job migrations.

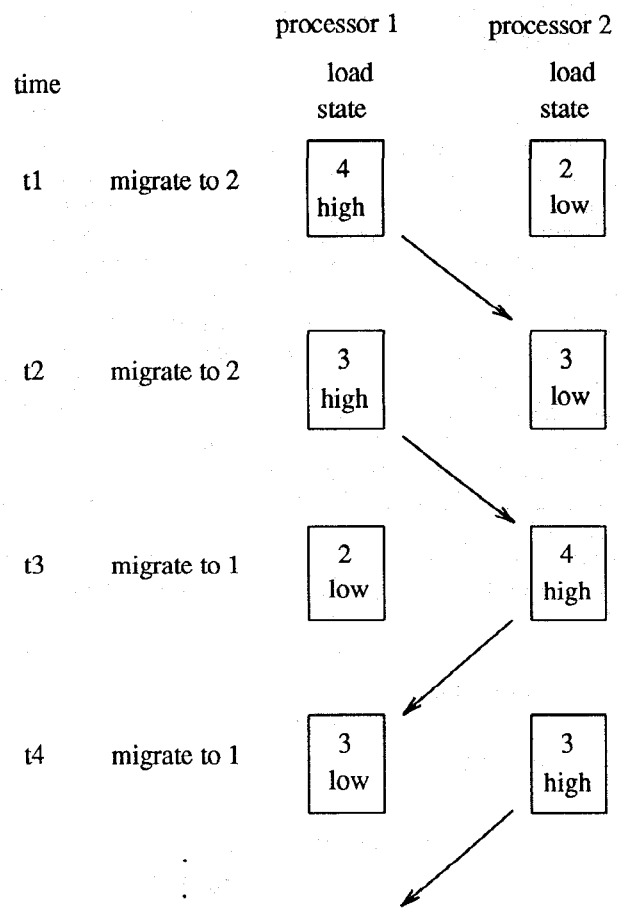

Figure 3. Thrashing

This would continue until a job finished or until another process could get in to receive a migrated job. Solutions to this could be to use a 3 stage load classification as is used by the drafting algorithm.

An important test is to compare the results of an execution where all the jobs are submitted at a single node, and every other node is in low state. From this, you can determine the effectiveness of the migration protocols. The system average load and average turnaround times are displayed in Figure 4. The program was run on 8 processing nodes for one hour with $\lambda = 2$ at one machine. Enough load was generated to keep all nodes active.

It should be quite obvious that under skewed load conditions the system performs better for each protocol over no migration. Even though the average load is only slightly better for the bidding program, over the entire test, there is a significant difference in the turnaround times. Figure 5 is a general comparison between bidding's and drafting's performance.

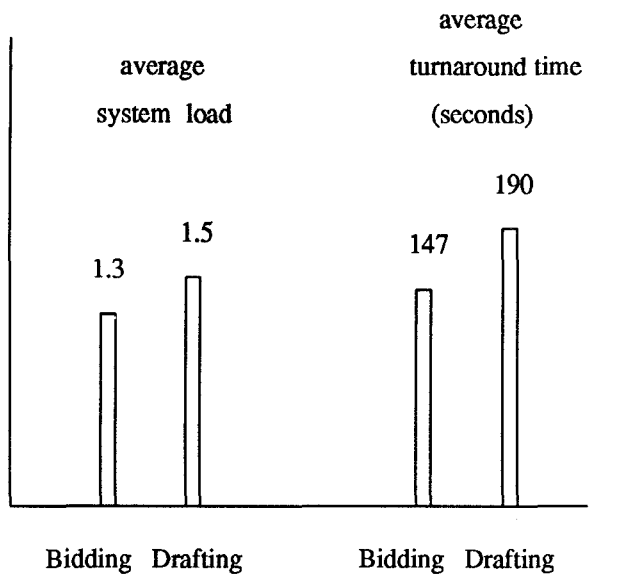

Figure 4. All jobs submitted to a single node

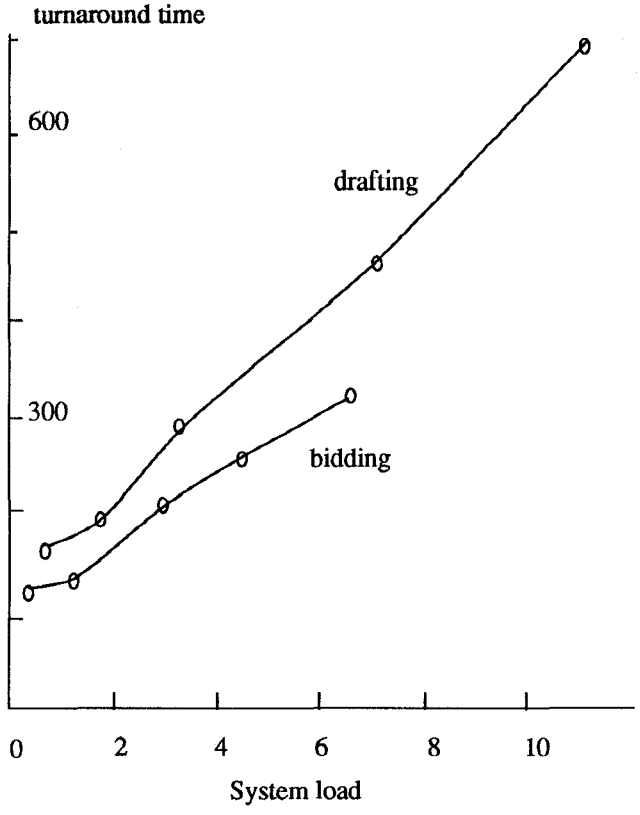

Figure 5. Performance Comparison: Job Turnaround Time vs. Load

4. CONCLUSIONS

In general, both algorithms produced an increase in system performance than with no migration, and bidding consistently showed better results in spite of process thrashing.

Drafting's main problems are timing and design structure. It takes 30 seconds from the time a migration condition occurs (existence of low and high load processors) to migrate a job. Bidding can migrate two jobs or more in a 15 second period. Also, in bidding, once one migration occurs the two nodes are in synchronization to migrate a second process. This added to its thrashing problem.

The bidding algorithm produced many more process migrations while the drafting algorithm transmitted mostly control messages. For this reason, the file server was of more benefit to the bidding program. In neither case did this communication load on the network have any noticeable effect on the system's overall performance. Thus, our results, as in [4] are highly dependent on the existence of the shared file server such that only job pointers, not jobs, are migrated.

A way to ensure that the first response is the best response is to require each responding processor to delay its bid by a multiple of their load measure. For example, three nodes are in low state and responding to a bid request. Their queue lengths are 1, 0, 1. For each job in the queue a processor waits 1 second before responding. Node B with 0 queue length would reply first, and the other nodes will reply 1 second later. Network delays would need to be considered.

Another possible refinement for both protocols would be to reduce the amount of *hand shaking*. Once a processor agrees to take some workload what is the point of double checking to make sure it can still accept the job? What if after you double check, the node becomes highly loaded? Maybe you should triple check... If migrated jobs were given a higher execution priority at there new node then even if that node became highly loaded the migrated job could still get better response at the new node than it did at its originating node.

In conclusion, these results match other preliminary results in that simple parameters achieve quite effective performance. To take it a step further, it's not as important to select the "best" job to be migrated to the "best" location as it is important to migrate a job quickly. The bidding protocol can migrate a job with 3 messages (bid request, response, migration) while drafting requires 4 sometimes 5 (load change to high, draft request, draft age response, draft select, migration). The first message is not needed if a process is already in high load. It is therefore believed that the bidding protocol will produce better overall system performance. A hybrid method with bidding's protocol, drafting's 3 stage classification, and

the first response refinement mentioned above might prove to give even better results. Of course, results could differ for each system, and extensive testing might prove other conclusions.

REFERENCES

[1] Stankovic, J.A. and I.S. Sidhu, "An Adaptive Bidding Algorithm For Processes, Clusters and Distributed Groups," Proc. 4th Int. Conf. Distributed Comput. Sys., pp. 49-59, 1984.

[2] Ni, L.M., Xu, C. and T.B. Gendreau, "A Distributed Drafting Algorithm For Load Balancing," IEEE Transactions on Software Engineering, Vol. SE-11, No. 10, pp. 1153-1161, Oct. 1985.

[3] Enslow, P.H., "What Is A Distributed Data Processing System?," Computer, pp. 13-21, Jan. 1978.

[4] Eager, D.L., "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing," Proc. ACM Sigmetrics Conference,1985, pp. 1-3.

[5] Chou, T.C.K. and J.A. Abraham, "Load Balancing In Distributed Systems," IEEE Transactions on Software Engineering, Vol. SE-8, No. 4, July, 1982.

[6] Hwang, K.,"A UNIX-based Local Computer Network With Load Balancing," Computer, pp. 55-64, Apr. 1982.

[7] Leffler, S., et.al., "The Design and Implemementation of The 4.3BSD Unix Operating System," Addison-Wesley 1989.

[8] Scheuermann, P. and G. Wu, "Broadcasting In Point-to-Point Computer Networks," Proc. 1984 Int. Conf. Parallel Processing, pp. 346-351, Aug. 1984.

[9] Bryant, R.M. and R.A. Finkel, "A Stable Distributed Scheduling Algorithm," Proc. 2nd Int. Conf. Distributed Comput. Sys., pp. 314-323, 1981.