In *Proceedings of the Int'l Conference on Parallel and Distributed Systems*, pages 104–109, Dijon, France, Sept. 1996.

The Influence of Resource Dependencies on Distributed Scheduling Policies for Load Sharing

Craig E. Wills
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA USA

Pete F. Bastien
M.I.T. Lincoln Laboratories
Lexington, MA USA

Abstract

Little research has been performed on the influence of resource dependencies on distributed scheduling policies for load sharing. Traditionally the CPU is the most contended resource, but other resources can be important in modern computing environments, due to the increasingly distributed nature of resources. For example, in an I/O intensive application, the number of I/O requests or the location in the distributed environment of the server for the requests may be important factors, which should be considered. This research examines the extent of resource dependencies in a typical distributed computing environment and how the resource dependencies can be taken into account to improve the performance of tasks selected for remote execution. Two distributed scheduling policies for load sharing that use resource dependencies are studied and are shown to improve performance.

KEYWORDS: load sharing, resource dependencies, distributed computing

1 Introduction

In networked environments the possibility of transferring work from heavily-loaded to lightly-loaded machines is attractive. In addition, the resources used by tasks executing in this environment are increasingly distributed among many machines within the environment. An executing task may use files served from different machines; with modern windowing environments, the display used to show the output of a task may not be local the machine executing the task.

Given the nature of modern computing environments, this work investigates the interaction between 1) load sharing—moving computations to lightly loaded hosts to increase response time and overall system performance; and 2) the increasingly distributed nature of resources in computing environments. This is an important area of work as we try to understand how the changing nature of computing environments affects how the traditional problem of load sharing should be approached.

Numerous distributed scheduling policies have been investigated for the problem of load sharing. Typically these policies have concentrated on a single resource, the CPU. However, the environment contains many networked resources such as file, display, name and print services. This work investigates which resources should be used and how to use them to improve the performance of load sharing policies.

The approach used for this work was to first understand the distributed nature of computations by gathering information about the location and amount of resources used by a task executing in a networked environment. A software tool was created to monitor executing tasks. The second part of the work was to examine the potential resources for consideration in a load sharing policy and to develop policies that took these resources into account. The final phase of the work was to compare the two policies we developed with a traditional load sharing policy to see if the performance of the executing tasks could be improved.

2 Related Work

In terms of predictability of process resource usage, one study has examined a scheme for predicting the CPU, file I/O, and memory usage of a process based on the identity of the program is developed [3]. This research is significant because it demonstrates that resource usage can be accurately predicted for a process based on past execution history.

The adaptive load sharing work by Eager, Lazowska and Zahorian is important work in showing that simple load sharing policies, with a small amount of state information can perform close to the expected perfor-

mance of ideal policies [4]. Three load sharing policies are examined with the Threshold policy serving as a basis in the current research. The question is whether other resources, rather than just the CPU considered in Eager, Lazowska, and Zahorian's work, need to be considered in load sharing for current computing environments.

Similarly, work by Zhou [6] shows that under realistic system loads and job mixes, a wide variety of load balancing schemes improve the system's response. The results of improved system performance for the case when a significant number of jobs are forced to be executed locally is encouraging. A problem with the research is that the effect of the cost of resources other than the CPU are not considered.

In terms of taking into account other resources in the load sharing selection process, the work by Wang, Krueger and Lui demonstrates that consideration of other resources besides just the compute resource can improve system response [5]. The work proposes a selection policy that uses a weight climbing algorithm and takes into account the CPU queue size, available memory and processor type of the local and remote hosts.

3 Resource Dependencies

To better understand the amount and location of resources used by a task executing in a distributed environment, we built a monitoring tool named *watch* [1, 2]. This tool was used to collect data on tasks executing in a distributed Unix environment consisting of workstations networked primarily by Ethernet.

Much data about the specific resources uses for a variety of commands were collected. In one example a single computation required resources from six distinct machines in order to carry out its task. This large number of resource dependencies, which is not atypical, motivates our research to examine their impact on load sharing decisions. Should they be considered in load sharing decisions for task placement and how does their inclusion compare with traditional load sharing policies examining only the CPU resource?

To test this question we gathered not only the location, but also the amount, of resource usage for typical tasks. This execution history was used to classify tasks according to the resources they use. Rather than examine all resources, our work concentrated on two types of resources—files and display. These resources were chosen because they are the most heavily used resources identified in our study. As a further distinction the file resources were classified into two types: system and user. System file resources include the executable file for a task along with task specific data files (such as font files) it uses. User file resources are referenced by a task for user-specific startup or are directly acted upon as part of the task. The file resources are distinguished because they be provided by different servers—it is not uncommon for a user's files to be served by one server with another server supplying executables.

4 Approach

Based on these findings, our work has explored the effects of resources other than the CPU in making load sharing decisions in a distributed environment. The file and display resources, along with the CPU resource, are the three resources studied in this work for their influence on the distributed load sharing decision.

To study the impact of these resources we decided to use the threshold load sharing policy of Eager, Lazowska and Zahorian [4] as our control policy. Their policy only takes into account the CPU load in making a load sharing decision, but is used because it has been shown to be relatively simple, yet effective as a load sharing policy.

As part of this work, two two other policies, which take into account the effect of file and display resources, are proposed. Details of all three policies, named ELZ Threshold (ELZThresh), Dependency Threshold (DependThresh) and Single Guided Probe (GuideThresh), are given in the following.

4.1 ELZ Threshold Policy

The ELZThresh policy is the threshold policy presented in [4]. It, along with the other policies we consider, has two parts: a transfer and a location decision. The transfer decision decides if a newly created task should be executed locally or an attempt to transfer the policy should be made. The location decision process determines where to transfer a task.

Each of these decisions is based upon a CPU queue length threshold, which is taken to be two for our work based on the results in [4]. When a task is created on a host, if the current CPU queue length is less than the threshold then the task is executed locally. Otherwise the location process is invoked to try and transfer the task to another host for execution. As part of this process, a target host is chosen at random and probed to determine its CPU queue length. If transferring the task to the target will not cause the target's queue

length to exceed the threshold then the task is transferred to the target. Otherwise a new host is randomly selected and the process is repeated. The number of probes is restricted by probe limit, which is set at three for our work. If the probe limit is reached then the transfer process fails and the task is executed locally on the originating host.

4.2 Dependency Threshold Policy

The DependThresh policy works similar to the ELZThresh policy in using a threshold-based policy with probes, but it includes file and display resource considerations in addition to the CPU. It assumes that values for file, display and CPU usage for a task can be predicted based on knowledge about the task. Each of these predicted resource usages is converted to a predicted time duration and summed to calculate a time value for the task. The threshold for the policy is then based upon time rather than CPU queue length.

When a task is created on a host its predicted time value is created based upon resource usage and location relative to the local host. If the predicted time value for the newly created task, added to the time values of the other tasks assigned to the local host would exceed the specified time threshold, then the newly create task is selected for a transfer attempt. In our work, this policy is studied with a time threshold of two times the mean predicted execution time of all tasks recorded in the execution history.

The location process for this policy is also similar to that for the ELZThresh policy, except that a time threshold rather than a CPU queue length threshold is used. Probes are randomly made to remote hosts with the decision to transfer to a remote host made based upon the amount and location of resources relative to the probed target host. The decision on whether to reject to accept the target is based on exceeding or not exceeding the target host's time threshold.

4.3 Guided Threshold Policy

The GuideThresh policy is motivated by the observation that it might be better to consider transfer of tasks to a host that already contains resources needed by the task. This approach would obviate remote access costs for such resources.

The transfer decision for the GuideThresh policy is identical to the DependThresh policy. The location process is similar, but rather than using random probes, the GuideThresh policy uses "guided" probes first to the user file server. If the user file server is also the originating host, then the system file server is selected as the first probe target. If both file servers are local then the first probe is random. If the first probe does not successfully locate a host then subsequent probes are random.

5 Simulation Study

In order to study these policies a simulation model was constructed based on the task and computing environment data we gathered. Simulation was used to best reflect an actual environment while providing us the opportunity to easily vary the parameters of the study.

5.1 System Model

The system model is based on a set of 20 networked computers, where all computers have two resources: CPU and display. The network is a broadcast communication network. Two types of computers are considered: hosts and file servers.

The hosts can be thought of as typical single processor workstations. Their purpose is to perform computations and provide display service to the users. The hosts may provide file service if the files are local to the host, but they do not normally provide file service to other hosts (unless a local task is transferred). All hosts are equivalent in terms of computational power.

File servers are identical to hosts, except they also contain disk resources, which are provided to other hosts. Depending on the simulation parameters, user and system file servers may or may not be configured. The absence of the file servers implies files are served by local hosts. File servers not only provide file service to other hosts, but also originate and perform computations.

5.2 Task Model

The input to the system model is a stream of tasks originating at the hosts. The rate is dependent on the system load of the simulation. The system load is the ratio of mean predicted execution time to the interarrival time of the tasks. Each task is considered to be individually schedulable, that is, there is no dependency relationships between the individual tasks. Each task uses some amount of CPU, file and display resources. Each of these resources is provided by hosts in the environment. The amount of each resource required by the tasks is used to describe the tasks and is based upon its execution history. The resource usage level are inputs to the load sharing policy indicating

the likely amount of resources needed by the application.

5.3 Base Simulation Parameters

As part of the simulation, system parameters were determined based upon our collected data. These parameters are shown in Table 1. The file access and transfer costs, as well as the display access and transfer costs vary depending upon whether these resources are local or remote. The transfer cost is the overhead cost in time of transferring a task to a remote host. The CPU usage is determined from the average CPU usage of the applications monitored in our environment. We assume that probes sent as part of the load sharing policy incur no time.

Table 1: System Parameters

Parameter	Local	Remote
File access cost	6.9 ms	13.1 ms
File transfer cost	1.2 KB/ms	0.4 KB/ms
Display access cost	0.96 ms	6.8 ms
Display transfer cost	2.9 KB/ms	0.15 KB/ms
Transfer cost	0	530 ms
CPU usage time	6.06 sec	6.06 sec

In actually setting up the simulation, it was determined that the job mix for the base simulation would be tasks with high disk and low display usage. Each task would expect to require a constant amount of CPU usage. To simplify the calculation of the time threshold, the file usage time and display usage times are calculated relative to the CPU usage time. These relative parameters are 0.5 and 0.1, respectively. These usages are only used in the determination of the time threshold.

6 Results

In order to determine the effectiveness of the proposed scheduling policies, the simulated system was run with various parameters beginning with the base parameters established in the previous section. The two major areas of modification are the environment and job mix.

The environment includes the location of the system and user files. The base simulation assumes that one system file server and a separate user file server serve the file resources in the system. A secondary aspect of the environment is the file access display access and the job transfer cost. The job mix is the composition of the jobs submitted to the system in terms of relative disk, display, and CPU use.

The following results show the response time (expected response time is normalized to one) versus the relative system load. The system load is the ratio of mean predicted execution time to the interarrival time of the tasks.

6.1 Base Case

The system response of the three policies using the base case parameters is shown in Figure 1. At lower system loads (less than 0.5) there is no significant difference in the response time of the policies. The GuideThresh and DependThresh policies exhibit an improved response time compared to the ELZThresh policy when the system load is moderate or high. At the highest system load simulated (0.95) the GuideThresh policy performs worse than the DependThresh policy. This worsening effect is caused by a low probability of the GuideThresh's first probe being successful.

Overall, the results show that at moderate system load the primary benefits of considering all resources occurs. At lower loads, the CPU is the primary consideration while at the highest loads there is little capacity in the network to make transfer of jobs worthwhile. The results show that in all cases, consideration of the file and display resources provides comparable or better results than when only the CPU is considered.

6.2 Response to Changes in the Environment

The first modification of the environment that is examined is the effect of the user's file location. In addition to user files located at a user file server as in the base case, we also examined an environment with all user files at the originating hosts.

With the user files located at originating hosts the system response for the DependThresh and GuideThresh policies is still better than the ELZThresh policy, but the improvement is not as significant as when the user files are located on a file server (Figure 1). In this case (user files at the originating host), the user file location acts to inhibit the transfer of a job since any remote execution of the job now requires remote file access as well as remote execution overhead and remote display access. It should be noted that in this case the GuideThresh policy selects the system file server as its first transfer target.

Figure 1: Base Case Simulation

The second modification to the environment that was examined is the effect on the system response for a change in the location of the system file location. The base case environment is to have the system files at a file server. The change in the system file location is the system files at the hosts performing the computation. In many systems, some of the system files are located at the site of computation (e.g. the */bin* and */etc* directory of Unix workstations). The results show little change from the previous system configurations.

In order to test the sensitivity of these results to the parameter costs we studied the effect of increasing the file, display and job transfer costs. The cost of file access for the base case (normal cost) is compared to a high file access cost specified at ten times the normal file access cost. The primary effect of the high file access cost is as a detriment to relocating the jobs on a remote host. Similar results were found for increasing the display and job transfer costs.

6.3 Response to Changes in the Job Mix

The final set of changes was to study a change in job mix—most specifically by studying applications which used more resources than the base case. In this case we changed the set of tasks to have an even higher file usage pattern. The results were as expected with the ELZThresh policy performing even worse (than the base case) relative to the other two policies. This response is due to the ELZThresh policy using only the job queue length as the determining factor in the transfer policy. The DependThresh policy includes the file loading as part of the transfer policy resulting in better transfer decisions. The GuideThresh policy performs better than the DependThresh policy since it tries to minimize the cost of file access by preferential probing the hosts that serve the files.

Similar results occur for the high display usage case. The other two policies perform better than the ELZThresh because they use the display usage in both the transfer and location decisions.

6.4 Summary

The major conclusion from these results is the time threshold concept improves the response time of the system at moderate to high system loads and when there is a significant file load. At low system loads or at light file loads the time threshold concept is similar to the traditional threshold policy.

In comparing the two new policies, DependThresh and GuideThresh, there is minimal difference in their response times in most cases. The exceptions occur at the highest system load value (0.95) where the GuideThresh performs worst than DependThresh and at very high disk usage where GuideThresh performs better than DependThresh.

Both the DependThresh and GuideThresh policies behaved consistently in response to changes in the environment, during the sensitivity analysis, and in response to changes in the job mix. The ELZThresh policy showed significant problems in the cases where resource are required, but not used in the scheduling policy's decisions. These cases are the high file load and high display load job mix cases where the DependThresh and GuideThresh policies improved the performance by 5 to 10%.

7 Future Work

The simulation of these policies has shown promise, but also points to directions of future work. Additional work could be to test the policies under different parameters and job mixes. Other resources could be considered, such as memory usage, although this resource is always co-resident with the CPU.

An obvious direction of any load sharing work is to implement it as part of a real system. Contributions to operating system development could be made by implementing and studying these load sharing policies in an actual computing environment.

Multimedia systems are an area which could benefit from the study of how dependencies could be used to improve system performance. The resources used by multimedia system are more varied than traditional systems, and the performance of multimedia systems rely on the efficient exploitation of the resources. For example, in traditional systems the requirements on display resources are limited and do not have a major time consideration, whereas in multimedia system the display media is much in demand and does have major time considerations.

Finally, the issue of scalability is not addressed in this research. It is an important consideration if the concepts of dependencies are to be used in a practical manner. If a load sharing policy is used to provide more efficient exploitation of the resource, then the size of the system and how well the policies scale to larger sized system is of critical importance.

8 Summary

In summary, the selection of which resources should be considered in a distributed scheduling policy and

how these resources should be used has been studied. The results shown that consideration of not only the compute resource, but also file and display resources can lead to improved performance over a traditional load sharing policy.

In general, the proposed policies improve performance in the cases where system load was moderate or high and where the display or file resources were in demand by the tasks under study. We believe that the consideration of the resource dependencies for a task can be done relatively easy when the execution history for a task is known. Overall, the results indicate that incorporation of dependencies shows promise. We will continue to work to understand how to make better use of resource dependency information.

References

[1] Pete Bastien. The influence of resource dependencies on load sharing distributed scheduling policies. Master's thesis, Computer Science Department, Worcester Polytechnic Institute, December 1995.

[2] David Brown, Craig Wills, Marton-Erno Balazs, and Peter Bastien. Computer network ease of service evaluation system. Report to Digital Equipment Corporation, March 1996.

[3] Murthy V. Devarakonda and Ravishankar Iyer. Predictability of process resource usage: A measurement-based study on unix. *IEEE Transactions on Software Engineering*, 15(12):1579–1586, December 1989.

[4] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, SE-12(5):662–675, May 1986.

[5] Chang-Jia Wang, Phillip Krueger, and Ming T. Liu. Intelligent job selection for distributed scheduling. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 517–524, 1993.

[6] Songnian Zhou. A trace-driven simulation study of dynamic load balancing. *IEEE Transactions on Software Engineering*, 14(9):1327–1341, September 1988.