Scheduling Resources in Multi-User, Heterogeneous, Computing Environments with SmartNet

Richard F. Freund*
Michael Gherrity*
Stephen Ambrosius*
Mark Campbell†
Mike Halderman*
Debra Hensgen‡
Elaine Keith†
Taylor Kidd‡
Matt Kussow†
John D. Lima†
Francesca Mirabile*
Lantz Moore§
Brad Rust†
H. J. Siegel¶

Abstract

It is increasingly common for computer users to have access to several computers on a network, and hence to be able to execute many of their tasks on any of several computers. The choice of which computers execute which tasks is commonly determined by users based on a knowledge of computer speeds for each task and the current load on each computer. A number of task scheduling systems have been developed that balance the load of the computers on the network, but such systems tend to minimize the idle time of the computers rather than minimize the idle time of the users. This paper focuses on the benefits that can be achieved when the scheduling system considers both the computer availabilities and the performance of each task on each computer. The SmartNet resource scheduling system is described and compared to two different resource allocation strategies: load balancing and user directed assignment. Results are presented where the operation of hundreds of different networks of computers running thousands of different mixes of tasks are simulated in a batch environment. These results indicate that, for the computer environments simulated, SmartNet outperforms both load balancing and user directed assignments, based on the maximum time users must wait for their tasks to finish.

*NCCOSC RDT&E Division (NRaD)
†Science Applications International Corporation
‡Naval Postgraduate School
§University of Cincinnati
¶Purdue University

1 Introduction

1.1 Overview

The computational resources available to an individual user may range from a personal computer to workstations to a variety of high-performance computers, all connected through combinations of local and wide area networks. In this distributed computing environment, the jobs of a single user are often affected by the jobs of other users, computer unavailability, and network congestion. It is not unusual for a user's jobs to be significantly delayed because of other jobs saturating network routes or sharing computational resources. Although in some cases this may be unavoidable, in many cases a user's jobs can be executed on alternate computers on the network. A resource manager with a global perspective of the network resources might be able to get the user's jobs completed in less time by executing these jobs on such alternate computers.

SmartNet is a resource scheduling system for distributed computing environments. It allows users to execute jobs on complex networks of different computers as if they were a single machine, or **metacomputer**. A user need not be concerned with the activi-

0-8186-8365-1/98 $10.00

ties of other users, nor even whether various machines in the metacomputer are temporarily unavailable. To allow users to continue with interfaces with which they are familiar, SmartNet can also function as a scheduling advisor to existing resource management tools [24].

SmartNet schedules and can manage the execution of each user's jobs in coordination with the other jobs in the metacomputer in a attempt to maximize the performance of the metacomputer for all users. From this global perspective, the emphasis is not on maximizing the efficient use of the machines in the metacomputer, but rather on maximizing the efficiency of the users of the metacomputer. The SmartNet performance metrics are based on how well the users are served, rather than on how well each machine is used.

Section 2 describes the current capabilities of the SmartNet system. This includes the ability to obey data dependencies between jobs, to account for the effect of different inputs on job execution times, and to use a variety of scheduling algorithms. In Section 3, the performance of a metacomputer from a global perspective is measured by the maximum amount of time any user must wait for jobs to finish. Using this metric and some basic assumptions regarding the use of a metacomputer (specified in Section 3), it is shown that for a network of heterogeneous machines, a scheduling system that considers both the machine availabilities and the affinity of jobs to machines significantly outperforms a scheduling system that attempts to balance the load across machines in the network, and also outperforms a system where users select the fastest machine to execute each of their jobs. Section 4 describes the future direction of SmartNet development.

1.2 Implementing Superconcurrency

The term **superconcurrency** [14, 15, 36] has been used to describe a technique for selecting the optimal suite of machines in a metacomputer to execute a given set of jobs [17]. To fully exploit the capabilities of a metacomputer, a single job must be decomposed into tasks, such that each task has relatively homogeneous computational requirements. These computational requirements are defined by the different machine architectures available in the metacomputer. This decomposition allows full exploitation of the fact that different tasks execute at different speeds on the different machines in the metacomputer.

Job decomposition can occur at several levels. In many cases, large projects are decomposed by developers into programs that can be executed separately by a computer operating system. Often each program is written to take advantage of the special computational abilities of a given computer architecture. Such programs can share data by reading and writing files. In what follows, an executable program will be called a **task**, and a project consisting of one or more tasks with data dependencies will be called a **job**. This paper focuses on the benefits that can be achieved when the full heterogeneity among the tasks in the the jobs and among the machines in the metacomputer is considered when scheduling the execution of the jobs on the metacomputer.

For some tasks, a finer level of decomposition may be beneficial. In this case a task may be decomposed into subtasks, each with homogeneous computational requirements [33]. Considerable effort has been applied to perform this decomposition automatically, but many open problems remain [34]. Several language extensions for parallel computation have been developed that allow this decomposition to be done by programmers. Systems that use such language features include AHS [8], HeNCE/PVM [2, 35], Legion [20], Mentat [19], and P4 [5]. These systems allow programmers to decompose tasks into subtasks to fully utilize the heterogeneous processing capabilities of a metacomputer. Although this paper concentrates on scheduling tasks on a metacomputer, scheduling subtasks on a metacomputer involves many of the same issues.

1.3 Task Scheduling on a Metacomputer with Multiple Users

How tasks are assigned to the machines of a metacomputer is an important factor that affects the performance of the metacomputer. The assignment of a task to a machine on which it executes slowly can significantly reduce overall performance. Likewise, resource contention must be considered when scheduling tasks on a metacomputer with multiple users. For example, an optimal assignment of tasks to idle machines can easily become suboptimal if one of the machines is suddenly loaded with a task from another user. There are several important issues that a scheduler for a network of heterogeneous machines with multiple users should consider.

Heterogeneity: A number of systems have been developed for managing the execution of tasks on a network of machines. Examples include Condor [4], OSF DCE [29], and PBS [22]. Such systems schedule tasks in order to evenly balance the load on the machines in the metacomputer [27]. Many of these load balancing schemes are modified in an attempt to account for differences in the capabilities between machines. One common method is to adjust the load on a machine based on its speed on a single task relative to the other

machines. However, this does not account for the different affinities between tasks and machines that occur as a result of heterogeneity [15].

Figure 1 shows a simple example where the schedule that minimizes the completion time of all the tasks of a given set of jobs (the **schedule length** [6]) distributes the load unevenly among three machines, actually leaving one of the machines in the metacomputer idle. Specifically, executing any task on machine 2 will cause the schedule length to be at least 8 time units, whereas if machine 2 is left idle a schedule length of 6 is possible. This shows the importance of considering task and machine affinities in scheduling tasks on a metacomputer.

The importance of considering heterogeneity is further demonstrated in table (a) of Figure 1 where machine 1 is four times as fast as machine 2 on every task, but the speed of machine 3 relative to machine 1 varies depending on the task it is executing. This is an appropriate model if, for example, machine 3 is a vector machine and some of the tasks are not vectorizable. Tasks that are vectorizable would tend to execute more quickly on the vector machine.

Figure 1 also shows that the optimal schedule does not assign task A to its best machine (machine 3) but rather to its second best machine (machine 1). If the user of task A assigns this task to machine 3, its best machine, then the other tasks will not finish until at least time 8. This illustrates that simply allowing users to assign their tasks to the machines that execute them fastest may not provide all users with the best performance of the metacomputer. A scheduler that considers all the tasks of the metacomputer may improve its performance for most users.

Task Execution Times: To exploit the heterogeneity among the tasks in the jobs and among the machines in a metacomputer, it is beneficial to estimate the execution time of each task on each machine. Such estimates can usually be made empirically, although some deterministic models have been developed [1, 38]. Section 3 presents simulation results showing that variations in the estimate of task execution times has a secondary effect on scheduler performance compared to the appropriate consideration of heterogeneity.

Of course, different input data can dramatically affect the execution times of many tasks, and this effect can vary depending on the machine executing the task. It is assumed the effect of input data on the execution time of a task on a given machine is deterministic. For example, the execution time of a Monte Carlo simulation task on a single processor machine might tend to be linearly related to the number of iterations specified in its input data. A scheduler can benefit by having access to both the characteristics of the input data that affect a task's execution time and an equation for how the task execution time can be estimated from these characteristics when executed on a given machine.

Network Usage: To fully exploit the capabilities of a metacomputer, it is not only beneficial to estimate the execution times of the tasks, but also to estimate the network traffic that occurs as tasks communicate. If the tasks of one user are heavily loading a network route between two machines, it may be beneficial to schedule other tasks on machines where a different network route can be used.

Sections 2 and 4 describe SmartNet's current and future approaches, respectively, to address these issues.

1.4 Relationships with Other Work

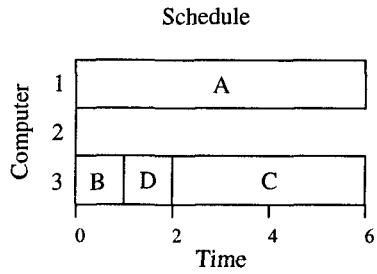
The importance of scheduling for the efficient use of distributed systems is well known, e.g., [10]. However, much of this research has been directed at scheduling the tasks of a single job on a network of processors, where there are no conflicting jobs or where the resource requirements of other jobs are unknown to the scheduler. In contrast, the SmartNet scheduler was designed to address multiple users competing for the resources of a metacomputer, and is most effective when the resource usage requirements of the jobs of each user can be estimated.

The importance of scheduling to achieve acceptable performance from a metacomputer with multiple users is also a major theme of the AppLeS [3] system. In this system, each task (called an application in AppLeS) would have its own AppLeS agent to select the metacomputer resources that will best meet that task's performance criteria. Because each agent schedules to maximize the performance of its associated task, the schedule that results would differ from the SmartNet schedule. As illustrated in Figure 1, SmartNet would attempt to minimize the longest execution time over all users. The emphasis in AppLeS is for each agent to optimize its own performance criteria rather than to cooperate with other agents to minimize a performance criteria shared by all the metacomputer users.

Task	Computer 1	2	3
A	6	24	5
B	2	8	1
C	8	32	4
D	2	8	1

Task Execution Times

(a)



(b)

Figure 1: An example where neither load balancing nor user directed assignments are the best scheduling strategies for a metacomputer. Table (a) gives the execution times of each task of a set of jobs on each machine. A Gantt chart of the schedule that minimizes the schedule length is shown in (b).

2 Current Implementation

2.1 Overview

The first version of SmartNet was operational in early 1994, and work has continued since then to increase its capabilities. The current version has proven to be useful in improving the performance of networks of heterogeneous machines, as will be shown in Section 3. Experience with this version has directed future efforts to enhance the system, and these will be described in Section 4.

Because batch systems such as Condor [4], OSF DCE [29], and PBS [22] perform many of the operations to manage the execution of scheduled tasks, this section will concentrate on the SmartNet scheduling capabilities not found in these systems. In fact, the SmartNet scheduling algorithms have been added to local versions of Condor [24] and Cray Research Inc.'s NQE.

2.2 How a Metacomputer and Tasks are Modeled in SmartNet

As described in Section 1.2, this paper will focus on scheduling executable tasks on a network of heterogeneous machines. The current version of Smartnet uses a **directed acyclic graph (DAG)** to specify data dependencies among the tasks of a job, and all the tasks of a single job communicate this data through files on a file server shared by all the tasks of the job. Different jobs may have different file servers. Communication among tasks can occur at the start or finish of a task. It is also assumed that tasks use a constant percentage of the processing and memory resources of the machine to which they are assigned, and that the executable program for each task is locally available to each machine which may execute the task.

It is assumed that the execution time of each task on each machine can be reasonably estimated. Task execution times are generally a function of a small number of input parameters, and better schedules result if both the values of these parameters and the appropriate time complexity function is available at the time the task is scheduled. For example, it may be possible to estimate the execution time of a task that contains a doubly nested loop by computing:

$$\text{execution time} = \alpha nm,$$

where α is a constant that may be found empirically, and n and m are the bounds on the loops. The user could provide this execution time formula to the SmartNet database for this task on the given machine, and the values of n and m could be given by the user when the task is submitted to the scheduler. SmartNet uses interpolation and extrapolation algorithms to extend rote learning of task execution times if either the parameters or the function are missing or only approximately correct.

The current version of SmartNet can account for latency and bandwidth between remote sites of the metacomputer, and also between the machines and a file server. Task priorities and data dependencies can be enforced by the scheduler. A background load on each machine and each network route can be considered when scheduling, however many complex network and processor contention issues are not considered in the current version. Although in many cases these network contention effects are negligible, there are problem domains where these effects are significant.

2.3 SmartNet Scheduling Algorithms

As shown in Figure 1, neither load balancing nor user directed assignment of resources may be as effective as a global scheduler for networks of heteroge-

neous machines. To most effectively schedule tasks on a metacomputer, the SmartNet scheduling heuristics all require estimates of the task execution times, and in some cases additional information about communication, memory, and processor usage. Estimates of task execution times are obtained through a combination of experiential information gathered automatically from trial runs, and, optionally, by time complexity equations (see Section 2.2) provided by the user.

In general, optimal multiprocessor scheduling is NP-complete [18], and hence SmartNet uses a variety of scheduling algorithms to attempt to obtain near-optimal schedules for different problems. The algorithms described below use a **deterministic execution simulation** [28] which performs a deterministic simulation of each task assuming the task execution times are equal to their estimated average values. A brief description of several of the SmartNet scheduling algorithms follows.

Maxmin and Minmin Algorithms:

If all the tasks to schedule are independent and compute intensive, then several of the algorithms described in [25] can be used. Specifically, algorithms D and E in [25] have been implemented and are called **maxmin** and **minmin** algorithms, respectively. Both have time complexities of $\mathcal{O}(mn^2)$, where m is the number of machines in the metacomputer and n is the number of tasks to schedule.

Both the maxmin and minmin algorithms consider a hypothetical assignment of tasks to machines, projecting when a machine will become idle based on the hypothetical assignment. Both algorithms determine, for each unassigned task, the earliest (minimum) time the task can be completed given the projected idle times of each machine and the estimated execution time of the task on each machine. The algorithms differ in their selection of which task to assign next given these minimum finish times. The maxmin algorithm selects the task that will take the maximum time to finish, whereas the minmin algorithm selects the task that could finish in the minimum time. Once selected, the task is assigned, the projected machine idle time is updated, and the task is removed from the set of unassigned tasks. The process is repeated until all tasks are assigned.

A schedule for 1000 tasks on a metacomputer of 100 machines can be determined in less than a minute using a typical workstation. Although [25] shows pathological problems where the resulting schedule length may be up to a factor of m worse than optimal, tests with environments such as those described in Section 3 have indicated that the schedule lengths are generally within 20% of optimal. Such tests were performed using simulations of small metacomputers and few tasks in order to perform an exhaustive search for an optimal schedule to be used for comparisons. In addition, for large metacomputers with many tasks, comparisons were possible against schedule lengths that were provably shorter than the optimal value [31].

Dependency Algorithms:

The following algorithms compute schedules when there are data dependencies between tasks. Although these tend to be the most frequently used SmartNet scheduling algorithms, a discussion of their effectiveness is outside the scope of this paper. A brief description is provided for completeness. For the experiments described in Section 3, there are no dependencies between tasks.

A Generational Algorithm:

This is a straightforward, cyclic method for mapping a set of dependent tasks onto available machines, that provides comparatively good schedules in a relatively short time [16]. During each cycle, a limited part of the scheduling problem is considered. Each task that has not satisfied all of its precedence constraints is considered ineligible for execution. All ineligible tasks are filtered out of the scheduling problem, forming a new smaller scheduling problem composed only of those tasks immediately eligible for execution. An auxiliary scheduling algorithm is then used to determine a schedule for the non-precedence-constrained problem. Upon detection of a rescheduling event, a new precedence-constrained scheduling problem is formed and the process repeats. One possible (indeed likely) rescheduling event is the completion of a previously scheduled task.

This generational algorithm is closely related to scheduling strategies such as Heaviest Node First scheduling [31] and Mapping Heuristic scheduling [9]. However this algorithm differs from these schedulers in that (1) all eligible tasks are rescheduled at each rescheduling event, and (2) the algorithm is designed to run dynamically as new task sets

are constantly added and completed.

A Clustering Algorithm:
A clustering algorithm [10] is provided for scheduling tasks with data dependencies that only use a portion of the available processing resources. For example, some tasks may consistently use only 50% of the CPU due to file I/O operations. In such cases, it is best to schedule several such tasks on a single machine (provided the machine has an operating system capable of multiprogramming [7], such as UNIX). In this case, the scheduler ensures that the memory requirements of the concurrent tasks are within the memory available on the machine, and that the concurrent tasks do not exceed the available network bandwidth.

Other Techniques:
A variety of experimental scheduling algorithms are also included in the current release of SmartNet. These include an algorithm using evolutionary programming [12], and another using a combination of genetic algorithms and simulated annealing [23, 32].

The added complexity of considering data dependencies in both the generational and clustering algorithms make these methods several times slower than the maxmin and minmin algorithms.

3 Simulation Results

3.1 Overview

It is difficult to precisely evaluate the performance of a scheduling algorithm for a metacomputer because this is dependent on many factors, such as the distribution of task execution times and characteristics of the metacomputer hardware. Tests using a specific metacomputer are not conclusive because the results are only valid for that metacomputer. Simulation studies provide the ability to demonstrate the effectiveness of a scheduling algorithm over a broad range of conditions, although there is no generally accepted set of benchmarks.

Although a large number of different metacomputers can be tested using simulation studies, there is an issue with the accuracy of the simulation. This section presents simulation studies where all tasks to be run are known initially (batch processing), tasks are independent, and network use is not significant, hence the need to accurately model specific features of a metacomputer for accurate simulation results are minimized. In addition, in this case, an assumption of single-programming (i.e., each machine executes a single task at a time) is appropriate to maximize the performance of the metacomputer.

Two simulation studies will be presented where a large number of randomly generated problems are scheduled. Section 3.2 describes simulations that were performed with results from the NAS parallel benchmarks [30] to model a metacomputer used in a typical production environment. Section 3.3 presents simulations that model a typical academic environment. Section 3.4 evaluates the effect of variations in the estimated task completion times for both the production and academic environments. As described in Section 2, there are SmartNet schedulers that consider communications between tasks, but the evaluation of these algorithms is outside the scope of this paper. The focus here is on demonstrating the benefits of considering heterogeneity in task scheduling. Additional information about SmartNet can be found in [13].

3.2 Simulating a Typical Production Environment

In the production environment modeled here, it is assumed that tasks tend to have long execution times. This would be typical of, say, batch jobs run overnight using a sizable metacomputer. To model this case, random sets of task execution times and machine speeds were generated using the NAS parallel benchmarks [30] as a template.

Specifically, ten machines were arbitrarily selected from the NAS database and the execution times of the eight NAS benchmarks on these machines (using the class A data size) were used. Table 1 shows the selected machines, and Table 2 shows the corresponding job execution times on each machine given in [30]. Notice the significant heterogeneity both in machine speeds for the same job (across rows) and job execution times for the same machine (across columns) shown in Table 2. No single machine is fastest on all the jobs, and the ratio of execution times on different machines is very much dependent on the job being executed.

Each test problem modeled a network of 20 machines with 100 tasks. The problems were randomly generated from Tables 1 and 2 as follows. Each of the 20 machines was selected by randomly picking one of the ten machines listed in Table 1 using a uniform distribution with replacement. Similarly, each of the 100 tasks was selected to correspond to one of the eight NAS jobs shown in Table 2. A complete 100×20 matrix of execution times was then created using the corresponding times in Table 2.

For these tests the maxmin algorithm was used by

index	Machine	Processors
1	HP/Convex Exemplar SPP2000	16
2	CRAY C90	16
3	CRAY T3E	256
4	CRAY Y-MP	8
5	DEC Alpha Server 8400 5/440 (437 MHz)	12
6	Fujitsu VPP700	32
7	IBM RS/6000 SP Thin-node2 (67 MHz)	128
8	Intel Paragon MP (SunMos turbo)	512
9	Kendell Square KSR2	64
10	SGI Origin2000 (195 MHz)	32

Table 1: The arbitrarily selected machines from the NAS database used to generate random test cases.

	Machine									
job	1	2	3	4	5	6	7	8	9	10
EP	10.5	2.36	0.4	15.87	8.46	1.03	1.31	0.87	13.0	3.55
MG	4.3	0.71	0.2	2.96	NA	0.25	0.63	1.36	5.7	2.12
CG	2.7	0.34	0.4	2.38	NA	0.67	1.48	NA	6.1	2.04
FT	NA	0.80	0.2	4.19	NA	0.33	1.30	1.92	6.5	3.16
IS	2.21	0.27	0.2	1.85	NA	0.98	0.61	2.29	3.9	1.21
LU	NA	10.17	4.2	49.5	67.97	10.06	15.9	NA	102.0	18.9
SP	56.6	12.82	6.0	64.6	91.43	4.53	20.6	NA	131.0	30.5
BT	55.3	20.3	6.2	114.0	103.5	4.93	20.8	113.0	130.0	30.0

Table 2: The execution times of each NAS job on the selected machines.

SmartNet, and both a load balancing algorithm and a user directed assignment algorithm were used for comparisons. The load balancing algorithm schedules each task on the machine that becomes idle first. Thus, load balancing considers machine availability, but ignores heterogeneity issues. The user directed assignment algorithm schedules each task on the machine that executes it fastest. Thus, user directed assignment considers heterogeneity, but ignores machine availability issues. The SmartNet maxmin algorithm, described in Section 2.3, considers both heterogeneity and machine availability.

Figure 2 shows the distribution of the ratio of the schedule length from the load balancing algorithm over the SmartNet schedule length for 1000 test problems. On average, the SmartNet schedule length is 36 times shorter than the load balance schedule length. Figure 3 shows the distribution of the ratio of the schedule length from the user directed assignment algorithm over the schedule length from SmartNet for 1000 test problems. On average, SmartNet performs two times better than the algorithm simulating user assignment.

3.3 Simulating a Typical Academic Environment

It has been shown [21] that the execution times of tasks are distributed exponentially in typical academic environments. This is quite different than the distribution of task execution times used in Section 3.2, which modeled a typical production environment using the NAS benchmarks. A similar series of tests as described in Section 3.2 were performed where this exponential distribution was used rather than the distribution based on the NAS benchmarks. Again, a batch submission of tasks is assumed.

Specifically, task execution times were randomly selected using an exponential distribution with a minimum of 10 and a mean of 1000. The results of 1000 samples from this distribution is shown in Figure 4. To provide some heterogeneity in the network, it was assumed that all machines had identical architectures but some were faster than others. Unlike the NAS case described in Section 3.2, the faster machines run all tasks faster than slower machines. The relative speeds of each of the 20 machines on the network were determined by randomly drawing from an exponential distribution with a minimum of 1 and a mean of 5.

It was found that the minmin algorithm was superior to the maxmin algorithm for this environment, and hence SmartNet was run using the minmin algorithm to schedule the tasks. Figure 5 shows the distribution of the ratio of the schedule length from the load balancing algorithm over the SmartNet schedule length for 1000 test problems. On average, the SmartNet schedule length is 2.2 times shorter than the load balance schedule length for this environment. Figure 6 shows the distribution of the ratio of the schedule length from the user directed assignment algorithm over the schedule length from SmartNet for 1000 test problems. On average, the SmartNet schedule length is 2.8 times shorter than the user assignment schedule length for this environment.

3.4 The Effects of Variations in Actual Task Execution Times on SmartNet Scheduling

The test results described in Sections 3.2 and 3.3 assume that the execution times of all the tasks on each machine were known prior to actually executing these tasks. The technique of load balancing is dynamic in that such information is not needed by the algorithm. When a task finally finishes, the machine on which it was running becomes idle and the next task in the queue can be started on that machine. In contrast, all the SmartNet schedulers assign all tasks to machines prior to the execution of any of these tasks.

To determine the sensitivity of the SmartNet schedulers to inaccurate estimates of task execution times, the tests described above were repeated with random noise added to the estimated task execution times. Specifically, after the SmartNet scheduler had determined the assignment of tasks to machines, the actual execution time of each task was determined by drawing a random time from a normal distribution with a mean of the estimated task execution time and a standard deviation a percent of the estimated task execution time. The load balancing algorithm used this actual task execution time rather than the estimated time used by the SmartNet scheduler.

Figure 7 shows a gradual increase in the load balance schedule length that results from adding random noise in this fashion to the task execution times. Although the average task execution time is not changed by adding this noise, the maximum execution time does increase. Because the load balancing scheduler occasionally assigns a task to the machine with one of these worst case times, and these bad assignments tend to dominate the schedule length, the schedule length can be seen in Figure 7 to increase as the noise level increases.

Figures 8 and 9 compare this dynamic load balancing schedule length with the SmartNet schedule length. Because the SmartNet schedulers consider heterogeneity, they are seen to be no more sensitive than a dynamic load balancing scheduler to this type of noise

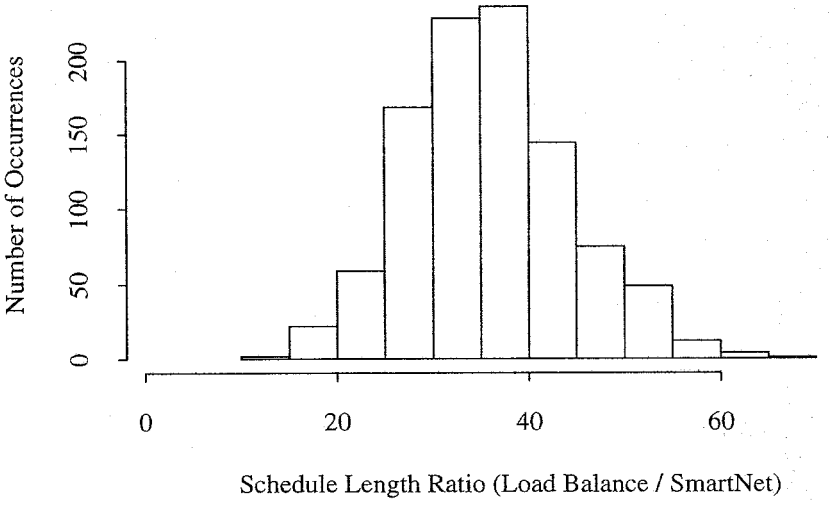


Figure 2: A comparison of the performance of a SmartNet scheduler to a load balancing scheduler using 1000 random problems modeling a typical production environment.

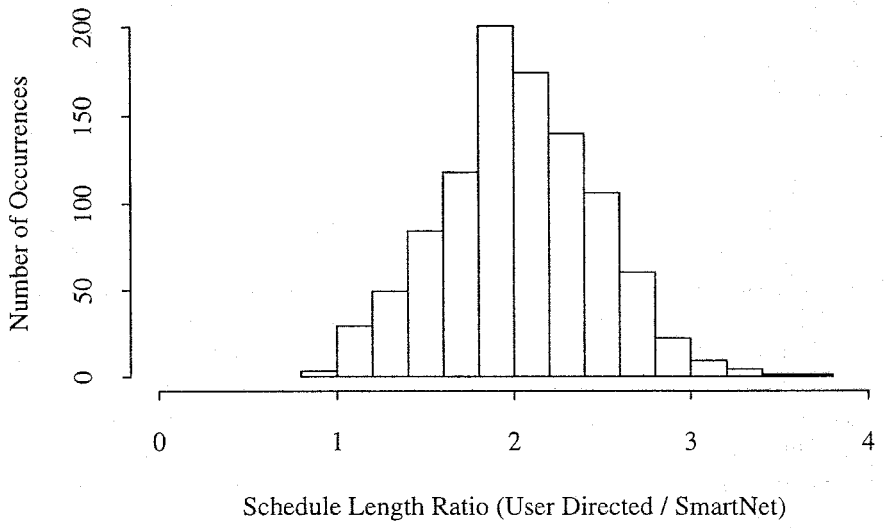


Figure 3: A comparison of the performance of a SmartNet scheduler to a scheduler simulating user directed assignment in a typical production environment.

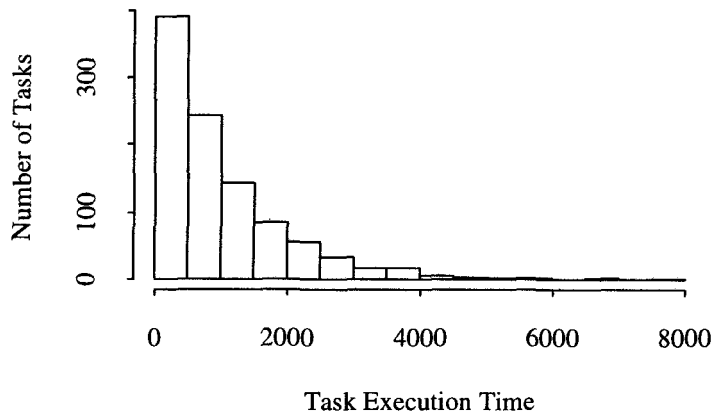


Figure 4: The distribution of task execution times used to simulate a typical academic environment. The results of 1000 random samples are shown.

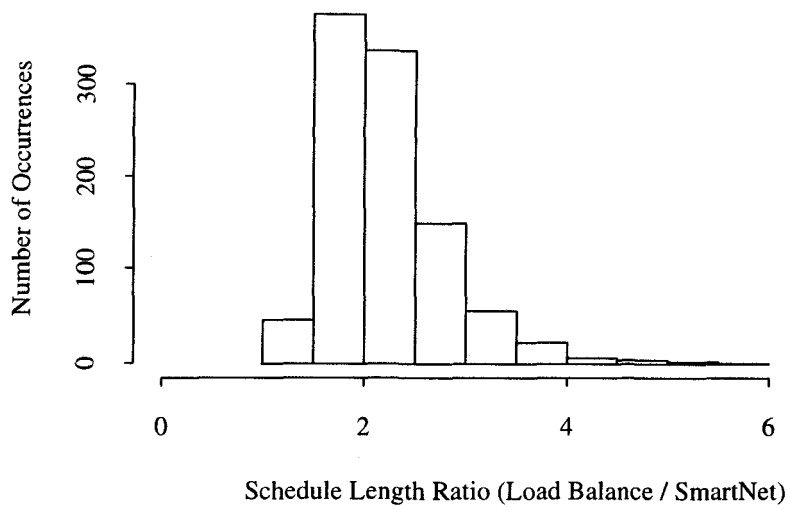


Figure 5: A comparison of the performance of a SmartNet scheduler to a load balancing scheduler using 1000 random problems modeling a typical academic environment with little heterogeneity.

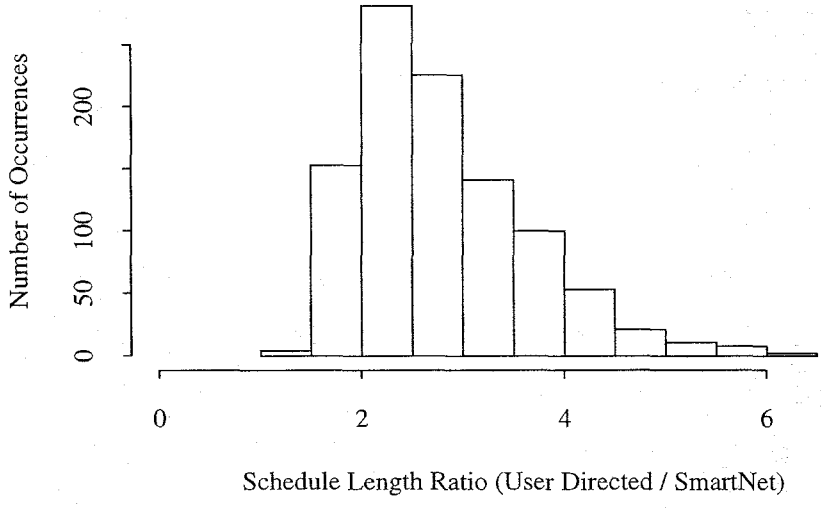


Figure 6: A comparison of the performance of a SmartNet scheduler to a scheduler simulating user directed assignment using 1000 random problems modeling a typical academic environment with little heterogeneity.

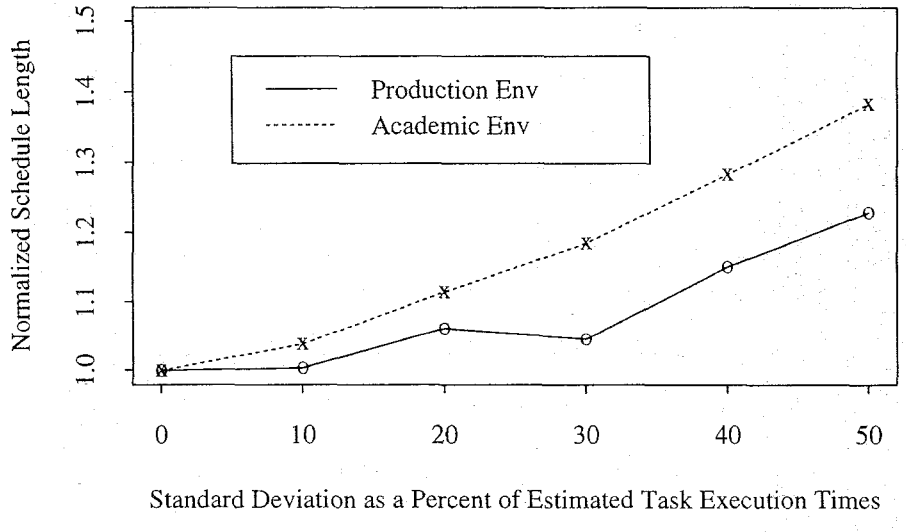


Figure 7: The performance of a dynamic load balancing scheduler when random noise is added to the task execution times. Each data point represents 100 random problems.

in the estimated task execution times. In fact, Figure 8 shows that the SmartNet scheduler is actually slightly better than the dynamic load balancing scheduler as the noise level is increased in this production environment.

4 The Future of Metacomputer Scheduling

Although an optimal solution to the task scheduling problem for a metacomputer is intractable, there are a number of polynomial time heuristic algorithms that provide solutions that are significantly better than merely balancing the load on all machines, or specifying a fixed assignment of tasks to machines. Section 3 presented several demonstrations of such an improvement in the simple case of tasks with no data dependencies or communication delays. Experiments using SmartNet to schedule problems where data dependencies and communication delays are included show that the improvements in the schedule length similar to those shown in Sections 3.2 and 3.4 is achieved, due to the benefit of scheduling tasks on the machines that run them best. Additional studies using SmartNet for scheduling tasks with data dependencies are documented in [26].

The SmartNet system has been operational since 1994 and is in its eighth release. Each release has significantly improved its ability to accurately model real-world computing environments. The development team has a Software Engineering Institute (SEI) level 3 rating, indicating the maturity and stability of the software. The system has been used for a variety of DARPA sponsored projects, as well as at the NIH, and is being tested for use at NASA. Section 2 has briefly outlined the current capabilities of the system. Future releases are planned to better account for networks and multiprogrammed machines. Specifically:

Network Routing:
A metacomputer can include numerous machines on several networks. Future versions of SmartNet are planned to better account for communication delays between tasks with data dependencies when tasks are scheduled on machines that require this communication to occur over several networks. Scheduling considering the contention effects of complex network routing has been a secondary concern due to increasing network speeds and the compute intensive nature of the applications that have been studied.

Resource Contention:
In many cases, it is appropriate to assume that a task controls a percentage of a resource throughout its entire execution time. Provided this portion of the resource is available, the task execution time is not affected by other tasks using other portions of the resource. This is the model used in the current SmartNet release. However it is also common that resources are shared among tasks, and such sharing causes the task execution times to be extended. This effect has been called **interference** in [37] and **slowdown** in [11]. Future releases of SmartNet will consider the effects of resource sharing on the execution time of tasks.

Optimization Criteria:
As described in Section 1, the SmartNet schedulers all attempt to minimize the schedule length. In some cases, notably for highly interactive environments, minimizing the average task finish time may be a more appropriate optimization criteria. It is planned that future releases of SmartNet will include schedulers that minimize this average task finish time (called **flow time** in [6]).

5 Conclusions

It has been shown in several cases that scheduling tasks considering both the machine availabilities and the heterogeneity of the machines in a metacomputer can increase the metacomputer performance. The objective of such scheduling is to minimize the total time users must wait for their tasks to finish.

Many scheduling approaches have been based on load balancing, which minimizes the idle time of the machines on the network rather than the idle time of the users of the network. In networks of homogeneous machines, the idle time of users can be minimized by minimizing the idle time of the machines. This is not the case in networks of heterogeneous machines. The schedulers used in the SmartNet system account for the heterogeneity of both the network machines and the user tasks. There are many factors that may create heterogeneity in what would appear to be a network of homogeneous machines, such as memory size differences between machines, network differences, or even differences in background loads.

The benefits of a global, centralized scheduler such as SmartNet diminish as the number of machines in the metacomputer grows large, due both to the time delays of the scheduling process itself, and to the associated increase in network traffic to and from the centralized scheduler. However, the simulation results described in Section 3 demonstrate conditions where a global scheduler like SmartNet is beneficial. For a metacomputer that covers the machines accessible to

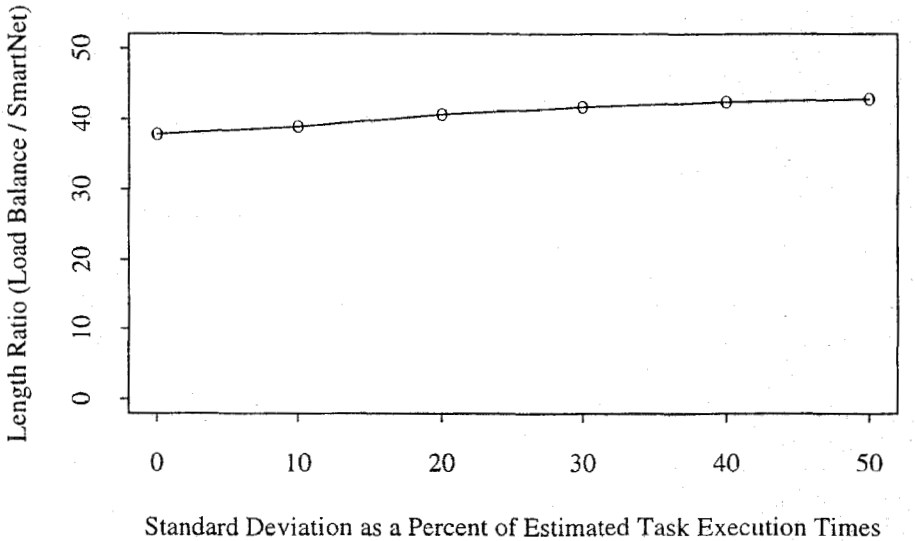


Figure 8: A comparison of the performance of a SmartNet scheduler to a load balancing scheduler when the estimated task execution times are not accurate. Each data point represents 100 random problems based on the NAS benchmarks.

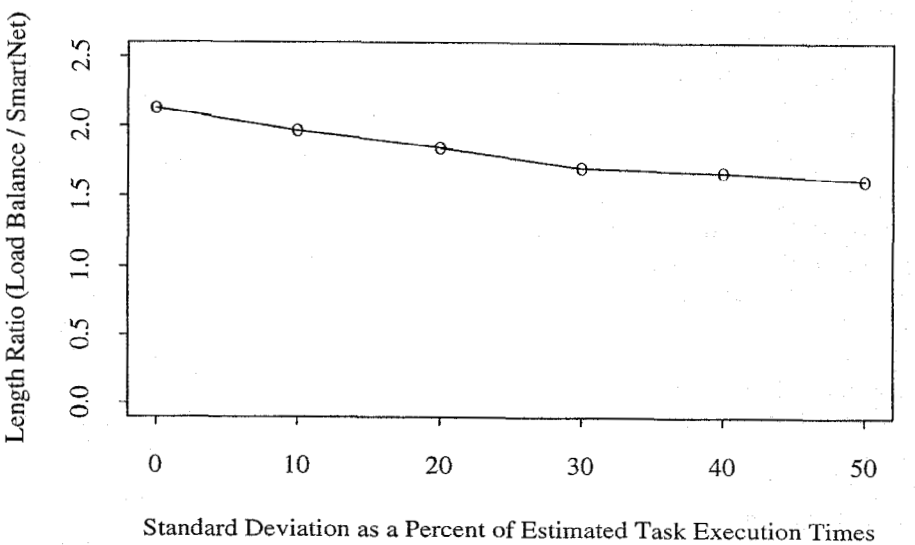


Figure 9: A comparison of the performance of a SmartNet scheduler to a load balancing scheduler when the estimated task execution times are not accurate. Each data point represents 100 random problems modeling a typical academic environment with little heterogeneity.

a typical department at a university, or a corporation, such global control is manageable. However, problems with scale clearly arise as the size of the metacomputer grows large [20]. Other tests, not described in this paper, indicate that the performance benefits of a global scheduler are still possible with metacomputers of several hundred machines when the task execution times are sufficiently long. For larger metacomputers, a hierarchy of SmartNet schedulers is currently being investigated.

Unlike many of the systems mentioned in Section 1, SmartNet does not constrain the user to a particular programming language, nor does it require the construction of special wrapper code for legacy programs. For best results, users need only provide a description of the time complexity of their tasks, and there are many tools that can help provide this information. By coordinating the execution time of user tasks, considering both machine availability and heterogeneity, the performance of a metacomputer may be substantially improved.

Acknowledgments

Portions of this work were supported by the NRaD Independent Research program, NASA, and DARPA. The authors thank the many people who have contributed to the SmartNet project, including Bill Adsit, Thomas Bayless, Michael Godfrey, Mitch Gregory, Joan Hammond, Roberta Hilton, Terry Koyama, Wanda Lam, Mary Lewis, Kathy Nolan, Sue Patterson, Bruce Rickard, Rod Roberts, Dave Schwarze, Dan Watson, Marc Weissman, and Bob Wellington.

References

[1] V. S. Adve. *Analyzing the Behavior and Performance of Parallel Systems.* PhD thesis, University of Wisconsin-Madison, December 1993.

[2] A. Beguelin, J. Dongarra, A. Geist, and R. Manchek. HeNCE: A heterogeneous network computing environment. *Scientific Programming*, 3(1):49–60, 1994.

[3] F. Berman and R. Wolski. Scheduling from the perspective of the application. In *The Fifth IEEE International Symposium on High Performance Distributed Computing*, August 1996.

[4] A. Bricker, M. Litzkow, and M. Livny. *Condor Technical Summary.* University of Wisconsin, Madison, version 4.1b edition, January 1992.

[5] R. M. Butler and E. L. Lusk. Monitors, messages, and clusters: the p4 parallel programming system. Technical report, Mathematics and Computer Science division, Argonne National Laboratory, Argonne, Illinois, 1992.

[6] E. G. Coffman, Jr. Introduction to deterministic scheduling theory. In E. G. Coffman, Jr., editor, *Computer and Job-Shop Scheduling Theory*, chapter 1, pages 1–50. John Wiley & Sons, New York, 1976.

[7] H. M. Deitel. *An Introduction to Operating Systems.* Addison-Wesley Publishing Co., Reading, Massachusetts, 1984.

[8] H. G. Dietz, W. E. Cohen, and B. K. Grant. Would you run it here... or there? (AHS: Automatic heterogeneous supercomputing). In *The 1993 International Conference on Parallel Processing*, volume II, pages 217–222, Saint Charles, Illinois, August 1993.

[9] H. El-Rewini and T. G. Lewis. Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel and Distributed Computing*, 9(2):138–153, 1990.

[10] H. El-Rewini, T. G. Lewis, and H. H. Ali. *Task Scheduling in Parallel and Distributed Systems.* PTR Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[11] S. M. Figueira and F. Berman. Modeling the effects of contention on the performance of heterogeneous applications. In *The High Performance Distributed Computing Conference*, 1996.

[12] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, New York, 1995.

[13] R. Freund, T. Kidd, D. Hensgen, and L. Moore. Smartnet: A scheduling framework for heterogeneous computing. In *The International Symposium on Parallel Architectures, Algorithms, and Networks*, Beijing, China, June 1996. IEEE Computer Society Press.

[14] R. F. Freund. Optimal selection theory for superconcurrency. In *Supercomputing '89*, pages 699–703, New York, NY, November 1989. ACM Press.

[15] R. F. Freund. SuperC or distributed heterogeneous HPC. *Computing Systems in Engineering*, 2(4):349–355, 1991.

[16] R. F. Freund, B. R. Carter, D. W. Watson, E. Keith, and F. Mirabile. Generational scheduling for heterogeneous computing systems. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 769–778, August 1996.

[17] R. F. Freund and H. J. Siegel. Heterogeneous processing. *Computer*, 26(6):13–17, June 1993.

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[19] A. S. Grimshaw, J. B. Weissman, and W. T. Strayer. Portable run-time support for dynamic object-oriented parallel processing. *ACM Transactions on Computer Systems*, 14(2):139–170, May 1996.

[20] A. S. Grimshaw, Wm. A. Wulf, and the Legion team. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1), January 1997.

[21] M. Harchol-Bulter and A. B. Downey. Exploiting process lifetime distributions for dynamic load balancing. Technical Report UCB/CSD-95-887, University of California, Berkeley, November 1995.

[22] R. L. Henderson and D. Tweten. *PBS: Portable Batch System requirements specification.* NASA Ames Research Center, April 1995.

[23] R. A. Henry, N. S. Flann, and D. W. Watson. A massively parallel SIMD algorithm for combinatorial optimization. In *The 1996 International Conference on Parallel Processing, vol. II*, pages 46–49, August 1996.

[24] D. A. Hensgen, L. Moore, T. Kidd, R. Freund, E. Keith, M. Kussow, J. Lima, and M. Campbell. Adding rescheduling to and integrating Condor with Smartnet. In *The 4th Heterogeneous Computing Workshop*, pages 4–12, April 1995.

[25] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the Association for Computing Machinery*, 24(2):280–289, April 1977.

[26] M. Janakiraman. Simulation results for heuristic algorithms for scheduling precedence-related tasks in heterogeneous environments. Master's thesis, University of Cincinnati, 1996.

[27] J. A. Kaplan and M. L. Nelson. A comparison of queueing, cluster and distributed computing systems. Technical Report NASA TM 109025, NASA Langley Research Center, June 1994.

[28] D. A. Menascé, D. Saha, S. C. Da Silva Porto, V. A. F. Almeida, and S. K. Tripathi. Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. *Journal of Parallel and Distributed Computing*, 28(1):1–18, 1995.

[29] Open Software Foundation, 11 Cambridge Center, Cambridge, Massachusetts. *Distributed Computing Environment: An overview*, January 1992.

[30] S. Saini and D. H. Bailey. NAS parallel benchmark (version 1.0) results 11-96. Technical Report NAS-96-18, NASA Ames Research Center, November 1996.

[31] B. Shirazi, M. Wang, and G. Pathak. Analysis and evaluation of heuristic methods for static task scheduling. *Journal of Parallel and Distributed Computing*, 10(3):222–232, November 1990.

[32] P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund. Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments. In *The 4th Heterogeneous Computing Workshop*, pages 98–104, April 1995.

[33] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li. Heterogeneous computing. In A. Y. Zomaya, editor, *Parallel and Distributed Computing Handbook*, chapter 25, pages 725–761. McGraw-Hill, New York, NY, 1996.

[34] H. J. Siegel, H. G. Dietz, and J. K. Antonio. Software support for heterogeneous computing. In A. B. Tucker, Jr., editor, *The Computer Science and Engineering Handbook.* CRC Press, Boca Raton, FL, 1997.

[35] V. S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2(4):315–339, December 1990.

[36] M-C. Wang, S-D. Kim, M. A. Nichols, R. F. Freund, H. J. Siegel, and W. G. Nation. Augmenting the optimal selection theory for superconcurrency. In *Workshop on Heterogeneous Processing*, pages 13–22, Los Alamitos, California, March 1992. IEEE Computer Society Press.

[37] J. Weissman. The interference paradigm for network job scheduling. In *The 5th Heterogeneous Computing Workshop*, pages 38–45. IEEE Computer Society Press, April 1996.

[38] Y. Yan, X. Zhang, and Y. Song. An effective and practical performance prediction model for parallel computing on non-dedicated heterogeneous NOW. *Journal of Parallel and Distributed Computing*, 38(1):63–80, October 1996.